ESCUELA POLITÉCNICA SUPERIOR

TRABAJO DE FIN DE GRADO

MOVIMIENTO EN ACIMUT Y ALTURA CONTROLADO POR GESTOS MEDIANTE UN ACELERÓMETRO CONECTADO A ARDUINO

Autor: Víctor Manuel SÁNCHEZ HERNÁNDEZ

Director: Guillermo ROBLES MUÑOZ

Madrid, Junio 2015

DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

GRADO EN INGENIERIA EN TECNOLOGIAS INDUSTRIALES

MOVIMIENTO EN ACIMUT Y ALTURA CONTROLADO POR GESTOS MEDIANTE UN ACELERÓMETRO CONECTADO A ARDUINO

Autor: Víctor Manuel SÁNCHEZ HERNÁNDEZ

Director: Guillermo ROBLES MUÑOZ

Madrid, Junio 2015

Título: Movimiento en acimut y altura controlado por gestos mediante un acelerómetro conectado a Arduino
Autor: Víctor Manuel Sánchez Hernández
Director: Guillermo Robles Muñoz

EL TRIBUNAL

Presidente:

Vocal:

Secretario:

Realizado el acto de defensa y lectura del Trabajo Fin de Grado el día ..... de ..... de ... en ....., en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de:

VOCAL

SECRETARIO PRESIDENTE

“Know where to find the information and how to use it. That’s the secret of success”

Albert Einstein

Agradecimientos

Quiero comenzar este breve apartado agradeciendo a mis amigos todo el apoyo recibido durante la realización de este trabajo. Comienzo por mis compañeros de piso, esos dos incansables que me han aguantado durante estos meses tan duros. Mis enfados, mi mal humor, mi ausencia aún estando en casa. Prosigo con aquellos que, a pesar de estar de exámenes, se han tomado la molestia de venir a visitarme, aunque fuera para mirar como trabajaba. Aunque no lo creáis, esa compañía vale más que mil palabras. A aquellos que no he podido visitar por estar ocupado, os agradezco vuestra paciencia y comprensión. Agradecer también a esas personas especiales que están lejos de mi, pero que con su cariño, amor y constante atención han sabido demostrarme su total apoyo, animándome en esos días duros de remo a contracorriente. Como olvidarme de mis amigos de la universidad, que no han parado de darme ánimos durante todo el proceso. Ese grupo de locos que hemos formado, grupo de amigos para toda la vida. Tenemos nuestras diferencias, soy consciente, pero nunca he visto un grupo tan unido como el nuestro. Ni Erasmus ni suspensos, nada ha conseguido separarnos. Juntos hemos aprendido a buscar nuestros límites, a dar lo mejor de nosotros mismos. Estoy muy orgulloso de vosotros chicos. Os lleváis los mejores cuatro años de mi vida.

Agradecer por supuesto a mis padres, que siempre me apoyan en todo con la única condición de que sea feliz. Gracias por llamarme, por entender que no me apetezca hablar, por aguantar que me pase meses sin pisar por casa. Al resto de mi familia: abuelos y tíos, que han confiado siempre en mi y que se que están orgullosos de lo que hemos conseguido. A mi hermana, que aunque no me hace mucho caso, se que me quiere y me valora y que siempre estará a mi lado.

Y por último, por supuesto, me gustaría agradecer a Guillermo la confianza depositada en mi. Porque un día aparecí desesperado por su despacho con la urgencia de acabar el grado este año, y decidió darme una oportunidad y asignarme un trabajo con el que he aprendido mucho sobre plataformas que desconocía: Arduino, Látex o Matlab. Por tu apoyo y comprensión, por ayudarme a cerrar contento un gran año, gracias Guillermo.

Índice general

Índice de figuras

Resumen

A lo largo de este documento se describe el estudio de diversas soluciones dirigidas a la regulación mediante gestos en altura y acimut de un dispositivo.

Debido a la existencia de posibles escenarios que imposibilitan el trabajo humano directo, existe la necesidad de controlar procesos de forma remota a través de una interfaz humano-máquina. Con el objetivo de hacer dicho control más ergonómico e intuitivo, se pretende realizar el mismo mediante gestos. En este caso concreto, para ilustrar un posible entorno de trabajo, se estudia el control remoto de la posición de una antena de medida de descargas parciales en aislantes eléctricos.

Para dicho fin se utilizará un microcontrolador Arduino en varias configuraciones constituyendo dos métodos distintos de adquisición de datos de posición. La primera configuración constará de un sensor capacitivo construido con planchas de papel de aluminio, modificando la posición de la antena en función de la variación de la capacidad percibida por dicho sensor. En la búsqueda de una solución más precisa y con una mayor probabilidad de implantación industrial, se valorará la utilización de un acelerómetro para llevar a cabo la función descrita con anterioridad. Ambas configuraciones se utilizarán para actuar sobre servomotores que componen el elemento de regulación.

Palabras clave: Arduino, gestos, instrumentación, acelerómetros, sensores capacitivos, interfaz humano-máquina.

Abstract

The present document describes the research made with the aim of the regulation of height and azimuth through gestures, taking into account diverse solutions.

Due to the existence of possible enviroments where human manual work is not possible, there is the need of remote control of processes through an man-machine interface. As a consequence, gesture control is introduced to make this operation more intuitive and convenient. This document in particular carries out the study of the adjustment of an measuring antenna for partial discharges inside electrical insulators.

In order to reach this goal, an Arduino microcontroller is used in different configurations, establishing two different methods of position acquiring. The first configuration includes a capacitive sensor based on aluminium films. Changes in capacitance are translated into position adjustments. Trying to achieve better accuracy and an easier industrial implementation, an accelerometer is used to detect this variety of gestures. Both configurations are used to control several servomotors which integrate the regulation method.

Keywords: Arduino, gestures, instrumentation, accelerometer, capacitive sensors, man-machine interface (MMI).

Capítulo 1

Introducción

1.1. Motivación

La idea original de este trabajo surgió de una propuesta planteada en la plataforma de financiación popular *Kickstarter*, llamada *"ArduSat - Your Arduino Experiment in Space"* [1] .

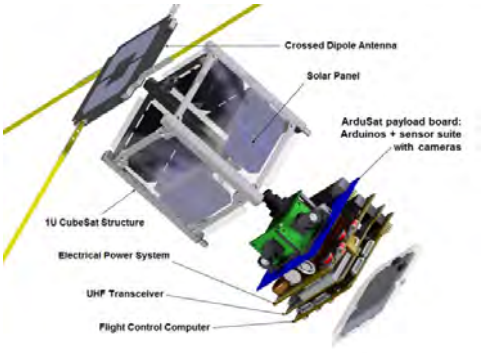

Figura 1.1: Estructura básica de Ardusat [1]

Se trata de una iniciativa generada por *"ppl4world"*, un grupo de profesionales relacionados con empresas del sector espacial y entusiastas de las plataformas de software libre. El proyecto propone la colocación en órbita de un satélite en miniatura, cuyo control completo se realiza a través de *Arduino*. Como añadido al "Crowdfunding", se solicitaba la creación de diversas partes del software del satélite con el objetivo de motivar la participación de la comunidad de software libre en su producción y lanzamiento.

Así es como nació éste trabajo. Se decidió comenzar a trabajar en la sección denominada *ENGINEERING: Your Eye in the Sky*. Dicha sección consistía en la escritura de una aplicación que sincronizara los movimientos de un acelerómetro conectado a un casco con el sistema de orientación de *Ardusat*, de forma que nuestros movimientos se transmitieran al satélite.

Sin embargo, reflexionando sobre las posibilidades del proyecto planteado se llegó a la conclusión de que cabía la posibilidad de aplicarlo como complemento al trabajo de fin de grado de unas compañeras del Grado en Ingeniería en Tecnologías

Industriales.

Ellas se encuentran desarrollando un programa dedicado a la medición de descargas parciales en conductores, para lo cuál necesitaban una antena de medición orientable. El fenómeno de las descargas parciales describe la avalancha de electrones que sucede en las vacuolas del interior de un aislante cuando éste está sometido a condiciones de alta tensión. Si las vacuolas son suficientemente grandes, éstas descargas pueden dañar el aislante debido a efectos de erosión y de producción de gases corrosivos. Una medida de estas descargas nos puede proporcionar información acerca del estado del aislante. Cuándo se produce una descarga parcial, ya que se trata de un fenómeno muy energético, se generan ondas de radiofrecuencia, de ultrasonidos, e incluso pueden generar luz y calor. Mediante diversos tipos de antenas se pueden medir dichas descargas, momento en el que entra en juego el desarrollo de este proyecto.

Pese a que la orientación de dicha antena se iba a hacer de forma automática, gracias a la aplicación desarrollada en este trabajo se posibilita su orientación mediante gestos sin exponernos a las condiciones de alta tensión en las que se generan dichas descargas. Actuando de forma directa sobre la orientación se podrá elegir el punto de medición, proporcionando al programa mayor versatilidad y precisión; y eliminando la dependencia de parámetros prefijados para la orientación automática.

1.2. Objetivos

Los objetivos principales del proyecto se resumen en:

- Comprender y aplicar las capacidades y el funcionamiento del microcontrolador *Arduino*, así como explorar las ilimitadas posibilidades que ofrece dicha plataforma de licencia en código abierto.
- Estudiar y construir un sensor capacitivo para la detección de posición, trabajar en su interacción con *Arduino* y actuar sobre un servomotor.
- Posicionar un dispositivo actuando sobre dos servomotores controlados por gestos. La interfaz de control está integrada por un acelerómetro conectado a *Arduino*.
- Construir un sistema de adquisición de datos de posición en *Matlab*, así como una representación gráfica del posicionamiento llevado a cabo.

1.3. Estado del arte

Hoy en día las habilidades de comunicación son vitales para la sociedad. Ya sea en entornos industriales o en actividades de la vida cotidiana existe una necesidad permanente del manejo apropiado de las habilidades comunicativas. Referida a actividades humano-humano, la comunicación se puede tornar complicada cuando uno de los interlocutores no conoce el lenguaje del otro. Este problema es aplicable a la interfaz humano-máquina. Cuando ambos no se comunican en el mismo lenguaje, es muy complicado que el receptor interprete con éxito el mensaje enviado por el emisor.

En un ambiente de trabajo puede darse la necesidad de comunicarse con una máquina de forma remota, segura, y no intrusiva. Estos requerimientos se consiguen cuando es posible simular la actividad humana normal mediante una representación virtual. En el caso particular de gestos humanos, es posible obtener una representación virtual del gesto y ser capaz de comunicarse en un entorno situado a gran distancia, ruidoso o con baja visibilidad.

Por los motivos expuestos, se han desarrollado una serie de algoritmos capaces de detectar e interpretar los gestos humanos.

1.3.1. Reconocimiento de gestos

El reconocimiento de gestos se refiere al proceso de entender y clasificar movimientos significativos realizados por el cuerpo humano: manos, brazos, cara e incluso cabeza. Se ha convertido recientemente en uno de los mayores campos de investigación, ya que es de gran importancia de cara al diseño de interfaces humano-máquina de inteligencia artificial para diversas aplicaciones, que comprenden desde el lenguaje de signos para la rehabilitación médica hasta aplicaciones de realidad virtual [2] .

Existen diversos tipos de sistemas de captura de gestos. Pueden tratarse de sistemas de captura óptica que utilizan visión virtual para el reconocimiento de la configuración de la mano; o pueden ser sistemas basados en la captura de datos mediante sensores unidos a la mano del usuario.
Un resumen de los principales sistemas de captura se describe a continuación [3] :

- *Marcadores de color*: Los marcadores de color para reconocimiento de gestos utilizan patrones de color para estimar la posición de la mano. La estimación se obtiene con cinemática inversa. Se ha demostrado que es posible construir un reconocimiento efectivo y de bajo coste con este enfoque, aunque precisa de una cámara debidamente posicionada y de buenas condiciones de luz.

Figura 1.2: Guantes que constituyen los marcadores de color, permitiendo el reconocimiento de la posición de las diferentes superficies de la mano [4]

- *Seguimiento de mano desnuda*: estos sistemas dependen de el reconocimiento de bordes y siluetas; y son generalmente robustos frente a las condiciones de iluminación. Su utilización concierne algoritmos que identifican el gesto de la mano en su espacio de movimientos. Se trata de un método costoso desde el punto de vista computacional y se aleja mucho de la respuesta en tiempo real, lo que lo convierte en no apto para interfaces humano-máquina.

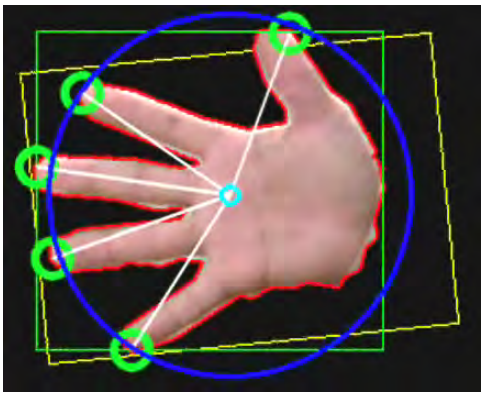

Figura 1.3: Visualización del seguimiento de la mano desnuda

- *Captura de movimientos basada en marcadores*: los sistemas basados en marcadores implican el uso de marcadores retroreflectivos o configuraciones multidispositivo de cámaras LED. Estos sistemas son muy precisos, pero a la vez costosos.

Figura 1.4: Marcadores retroreflectivos colocados sobre la mano [5]

- *Estimación de la posición a partir de datos*: este sistema hace uso de los valores recibidos desde los sensores unidos a la mano para definir la posición de ésta. Este enfoque permite un cálculo simple para estimar la posición de la mano. Sin embargo, este tipo de sistema se puede volver muy intrusivo con facilidad ya que necesita que el usuario se integre con los sensores, como por ejemplo, con la vestimenta.

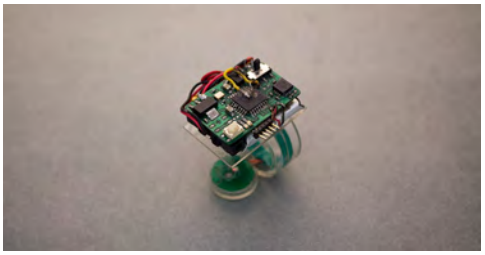

Figura 1.5: Anillo con reconocimiento de gestos y medición de la actividad © 2014 Dhruv Saxena

1.3.2. Reconocimiento de gestos con acelerómetros

La proliferación de la tecnología, y específicamente de la microelectrónica, ha inspirado la investigación en el campo del reconocimiento de gestos basado en acelerómetros. Éstos ya han sido incluidos en diversos dispositivos electrónicos, generalmente de uso personal, como teléfonos inteligentes o tabletas.

Su utilización se ha vuelto muy común en las aplicaciones tecnológicas de hoy en día gracias a su coste reducido y su simplicidad de funcionamiento. Generalmente se combinan con otros dispositivos como giroscopios de dos ejes[1] o sensores EMG[2] para mejorar el rendimiento del sistema.

En este trabajo se utilizará un sistema basado en un acelerómetro de tres ejes, ya que por motivos de coste, y dado que nuestra aplicación no precisa de altos volúmenes de información, no alcanzará un nivel de intrusión demasiado alto.

Así mismo se evaluará un prototipo constituido por un sensor capacitivo, de forma que se disponga una base desde la que comparar ambas configuraciones.

[1] Un giroscopio es un dispositivo que mide o mantienen el movimiento de rotación, a través de la medida de la velocidad angular.

[2] Un sensor de electromiografía mide la señal mioeléctrica, aquella que se produce cuando ocurre la contracción muscular [6] .

1.4. Entorno económico

Dado que el presente trabajo está definido como proyecto de investigación, no se considera relevante la inclusión de un presupuesto.

El coste de los componentes utilizados ha sido cubierto por la Universidad Carlos III de Madrid. También se a aprovechado la posesión personal de cientras herramientas (Arduino), que se han utilizado a lo largo del desarrollo del trabajo.

Los documentos citados en este documento han sido proporcionados por la Biblioteca de la Universidad Carlos III de Madrid.

1.5. Estructura del documento

Este documento se ha estructurado en cuatro partes principales:

- Para comenzar, se incluye un breve resumen de la motivación y los objetivos que han tomado parte en este proyecto, así como el estado del arte en el ámbito tratado.
- Tras ésto, se procede a la descripción las herramientas que se han utilizado para desarrollar el trabajo, en un capítulo denominado 'Detalle de herramientas utilizadas'.
- En la tercera parte se describen los métodos desarrollados durante la evolución de este trabajo. Para ello, se realiza una introducción de los conceptos teóricos en los que se ha basado el método. Así mismo se incluyen los circuitos construidos para la consecución de los objetivos y la explicación de las partes más significativas del código escrito. Por último, se expondrán las conclusiones y se analizará la adecuación de los métodos al cumplimiento de los objetivos.
- Para finalizar, en el último capítulo se realizan unas conclusiones generales y se incluye una sección de trabajos futuros donde se describen los posibles desarrollos y los trabajos en potencia de este proyecto.

Al final del documento se encuentran los apéndices, donde se ha incluido todo el código utilizado y otra información de remarcable importancia.

Capítulo 2

Detalle de herramientas utilizadas

En este capítulo se detallan los componentes de hardware y software utilizados en ambas configuraciones de medición. Mas adelante se detallará cada método de medición de forma concreta.

2.1. Arduino

2.1.1. Introducción a Arduino

Arduino es una herramienta destinada a acercar el mundo de la microelectrónica a aquellas personas que, a pesar de no ser expertos, tienen curiosidad por explorar los métodos de la programación. Además, está diseñada de manera que permite interactuar con el entorno de forma directa, como por ejemplo, haciendo medidas de temperatura o de movimiento de una forma sencilla e intuitiva. La herramienta está constituida por un microcontrolador montado sobre una placa con diversos conectores que permiten su interacción con variables externas. Así mismo, incluye su propio entorno de desarrollo que permite escribir software para la placa.

Arduino se puede utilizar para el desarrollo de objetos interactivos, adquiriendo datos de entrada a través de una amplia variedad de interruptores y sensores, y controlando luces, motores y otros dispositivos de salida. Los proyectos de *Arduino* poseen la capacidad de ser independientes y autónomos, o pueden comunicarse con software que se esté ejecutando en el ordenador (como código Flash o Java.) Las placas se pueden ensamblar a mano o pueden ser compradas preensambladas, y el entorno de desarrollo de código abierto *IDE* (Integrated Development Environment) se puede descargar de forma gratuita.

El entrono de programación de *Arduino* utiliza su propio lenguaje, igualmente diseñado en busca de la simplicidad y facilidad de aprendizaje. Es muy similar a C++.

2.1.2. El objetivo de Arduino

A pesar de que existen numerosos ejemplos de plataformas que permiten la comunicación de los ordenadores con el exterior, *Arduino* pretende simplificar los

procedimientos de programación de otras plataformas, pero ofreciendo los mismos servicios en un paquete fácil de usar.

El objetivo final de esta iniciativa es hacer el uso de los microcontroladores mas sencillo y accesible, objetivo que cumple ofreciendo las siguientes ventajas [7] :

- *Bajo coste*: las placas *Arduino* son mucho más baratas que otras plataformas de microcontroladores, e incluso pueden ser ensambladas a mano.
- *Software multiplataforma*: Los programas para *Arduino* pueden ser desarrollados en Windows, Macintosh OS y Linux.
- *Entorno de programación simple*: el entorno *IDE* es fácil de usar para principiantes, pero lo suficientemente flexible como para que los usuarios avanzados puedan aprovechar todo su potencial. Además, como se ha indicado anteriormente, es gratuito y de código abierto.
- *Software de código abierto y con capacidad de ampliación*: Todo el software de *Arduino* se publica de forma abierta. Así mismo, el lenguaje puede ser ampliado por medio de la utilización de librerías de C++.
- *Hardware ampliable*: los planos y esquemas de las placas han sido publicados bajo una licencia Creative Commons, por lo que diseñadores de circuitos con experiencia pueden modificar y mejorar la circuitería de las placas.

2.1.3. Tipos de placas Arduino

En función de la aplicación buscada, existen diversas placas preensambladas, que difieren en forma, rendimiento y prestaciones.

Se pueden encontrar desde placas básicas como *Arduino Uno*, hasta placas diseñadas para prendas y textiles como *Arduino LilyPad*. En la figura 2.1 se pueden apreciar algunos modelos disponibles actualmente [7] .

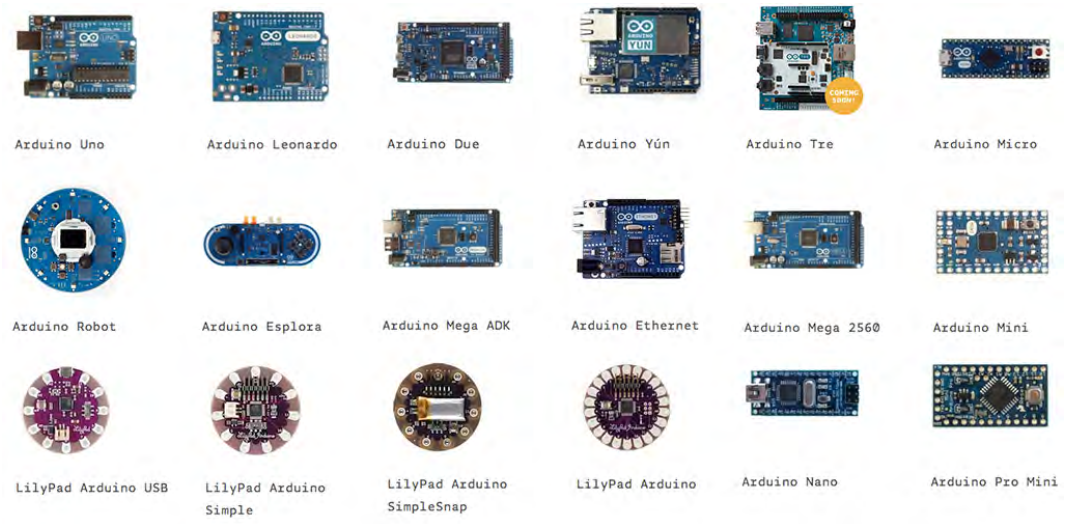

Figura 2.1: Algunas de las placas *Arduino* actuales [7]

Una amplia comparativa de los diferentes modelos preensamblados se puede encontrar en 'Comparativa de las placas *Arduino*'.

Para este trabajo se ha seleccionado *Arduino Uno*, ya que es la placa más asequible que ofrece una versatilidad suficiente como para alcanzar los objetivos planteados.

2.1.4. Arduino Uno

Arduino Uno es una placa basada en el microcontrolador ATmega328. Posee 14 entradas y salidas digitales (6 de los cuales pueden usarse como salidas PWM[1]), 6 entradas analógicas, conexión usb, conector de alimentación, y un botón de reseteado. Posee todo lo necesario para alimentar y dar soporte al microcontrolador.

Sus características principales se resumen en la siguiente tabla:

Microprocesador	ATmega328
Tensión de operación	5V
Tensión de entrada (recomendado)	7-12V
Tensión de entrada (límites)	6-20V
Pins I/O Digitales	14 (de los cuáles 6 con capacidad PWM)
Pins de entrada analógica	6
Corriente DC por pin I/O	40 mA
Corriente DC por pin de 3.3V	50 mA
Memoria Flash	32 KB (ATmega328) 0.5 KB para el bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Velocidad del reloj	16 MHz
Longitud	68.6 mm
Anchura	53.4 mm
Peso	25 g

La versión utilizada en concreto es la revisión número tres, por tanto: *Arduino Uno Rev3*.

[1]Pulse Width Modulation: técnica que permite codificar un mensaje dentro de una señal pulsada. Se explica de forma detallada más adelante

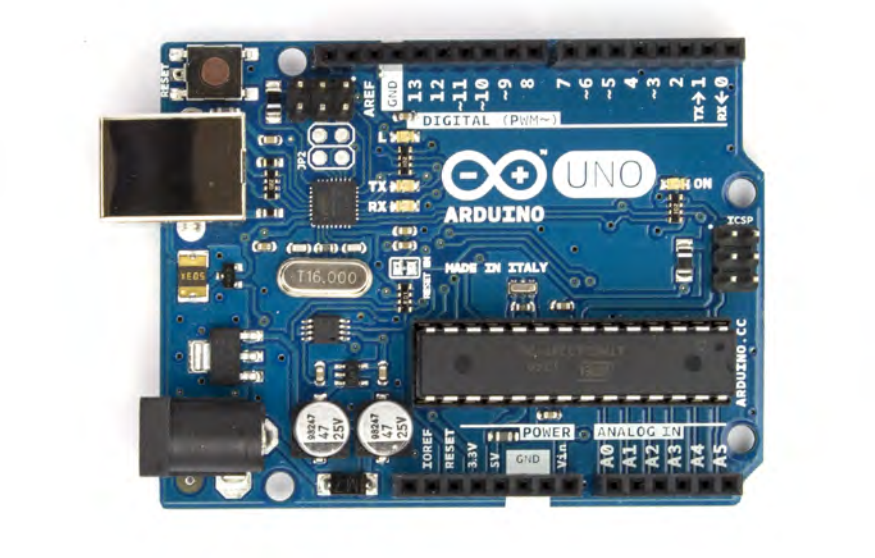

Figura 2.2: Imagen frontal de *Arduino Uno Rev3* [7]

2.1.4.1. Alimentación

Arduino Uno puede ser alimentado a través de la conexión USB o con una fuente de alimentación externa. El método de alimentación se elige de forma automática.

La placa tiene unos límites de alimentación comprendidos entre 6 y 20 voltios. Sin embargo, si se alimenta con menos de 7 voltios, el pin de 5V puede proporcionar menos de 5 voltios y la placa puede ser inestable. Si la placa se alimenta con más de 12 voltios, el regulador de voltaje se puede sobrecalentar y dañar el microcontrolador. Consecuentemente, el voltaje recomendado oscila entre 7 y 12 voltios.

La alimentación externa (no USB) puede proceder de un adaptador AC-DC o de una batería. La batería se puede acoplar mediante los pines *Gnd* y *Vin* del conector *POWER*.

Figura 2.3: USB y *Power Jack*

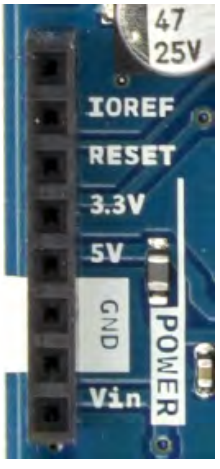

Figura 2.4: Conector *POWER*

El único pin del conector *POWER* cuya utilización no se intuye de forma inmediata es el denominado *IOREF*. Proporciona a l placa *Arduino* una referencia de voltaje con la que el microcontrolador opera.

2.1.4.2. Memoria

El ATmega328 posee 32KB de memoria, de los cuales 0.5KB son utilizados por el *bootloader*. Así mismo está equipado con con 2KB de *SRAM* y 1KB de *EEPROM* (que puede ser leída y escrita gracias a la librería *EEPROM*).

2.1.4.3. Entradas y salidas

Cada uno de los 14 pines digitales pueden ser configurados como entradas o salidas mediante las funciones *pinMode(), digitalWrite()* y *digitalRead()*. Operan a una tensión de 5 voltios, y pueden recibir o proporcionar un máximo de 40mA.
Algunos pines llevan a cabo funciones especiales:

- *Serial: 0(RX) y 1(TX)*. Son usados para recibir (RX) y transmitir (TX) datos serie TTL[2]. Se utilizarán estos puertos para cargar programas en la placa e interactuar con el monitor serie.
- *Interruptores externos: 2 y 3*. Estos pines se pueden configurar para disparar un interruptor ante una entrada de nivel bajo, ante un flanco de bajada o subida; o ante el cambio de un valor.
- *PWM: 3,5,6,9,10 y 11*. Proporcionan una salida PWM de 8 bits con la función *analogWrite()*. Con esta tecnología se consigue transmitir la información angular a los servomotores.
- *SPI: 10(SS), 11(MOSI), 12(MISO), 13(SCK)*. Estos pines facilitan la comunicación de diversos dispositivos utilizando la librería *SPI*. Se utilizarán estas funciones para comunicarnos con el acelerómetro y obtener los datos de orientación.
- *LED: 13*. Este pin está conectado a un led incrustado en la placa *Arduino*.

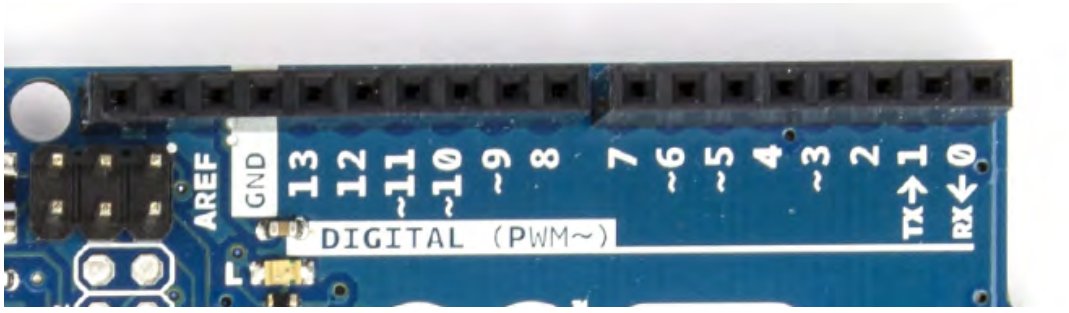

Figura 2.5: Pines digitales de *Arduino Uno*

[2]Comunicación serie con lógica transistor-transistor. La comunicación serie TTL siempre se mantiene entre los límites 0V y *VCC*, que es comúnmente 5V o 3.3V.

La placa también posee 6 entradas analógicas, nombradas desde A0 hasta A5, cada uno de los cuáles proporciona 10 bits de resolución (1024 valores diferentes). Estos puertos miden por defecto voltajes entre 0V y 5V, aunque el límite superior se puede modificar usando el pin *AREF* y la función *analogReference()*. De nuevo, algunas entradas desempeñan funciones especializadas:

- *TWI: pines A4 o SDA y pines A5 o SCL*. Soportan comunicación TWI[3] mediante la librería *Wire*.

Figura 2.6: Pines analógicos de *Arduino Uno*

Por último, existen dos puertos adicionales:

- *AREF*: mencionado anteriormente, constituye la referencia para el límite de voltaje superior de los pines analógicos.
- *Reset*: cuando es accionado, resetea el microcontrolador.

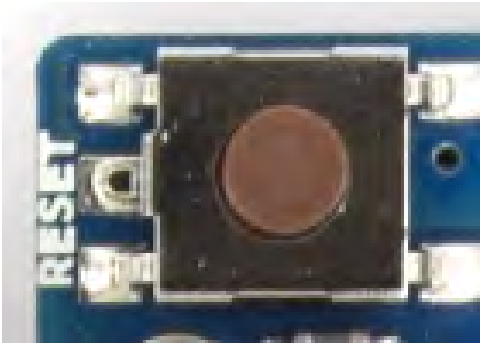

Figura 2.7: Botón de reseteo de *Arduino Uno*

2.1.5. IDE, el entorno de desarrollo Arduino

El *IDE* de *Arduino* (Integrated Development Environment) es el entorno que se utiliza para escribir código compatible con las placas *Arduino*, así como para verificarlo y transmitirlo.

La interfaz consta de un editor de texto, un área de mensajes, una consola de texto y una barra de tareas con las funciones más comunes.

[3]Se trata de un protocolo de comunicación a través de un bus I^2C o SPI. 2.4.3.3.

Figura 2.8: Interfaz del entorno de desarrollo *Arduino*

El software escrito mediante la utilización del *IDE* de *Arduino* se denomina 'sketch', que es escrito en el editor de texto. Estos documentos son guardados con una extensión '.ino'. El área de mensajes proporciona información durante los procesos de guardado y exportación, así como mensajes de error. La consola de texto muestra la salida de texto del entorno *Arduino* incluyendo diversa información, como por ejemplo, los mensajes completos de error.

En la barra de tareas se encuentran los botones que nos permitirán verificar y transmitir nuestros programas, crear, abrir, y guardar 'sketches', y abrir el monitor serie.

2.1.5.1. Monitor serie

Se trata de una herramienta muy importante que proporciona el entorno de desarrollo de *Arduino*. Nos permitirá comunicarnos con la placa mediante el puerto serie se estén ejecutando programas no autónomos.

Posee una interfaz muy sencilla, que nos permite ver mensajes generados por el microcontrolador, así como enviar datos o comandos que hayan sido implementados en nuestro programa a través del botón *Send*. Así mismo nos permite seleccionar la velocidad de bits a la que transmitirá el puerto serie. Por defecto es de 9600 baudios. Esto significa que la velocidad de transmisión será de 1200 bytes por segundo (9600 bits / 8 bits por byte).

Para poder establecer una comunicación con la placa a través del monitor serie, es imprescindible la utilización de la función *Serial.begin()*. Así mismo, la velocidad de transmisión debe ser la misma, tanto la especificada en el sistema como la seleccionada en el monitor serie.

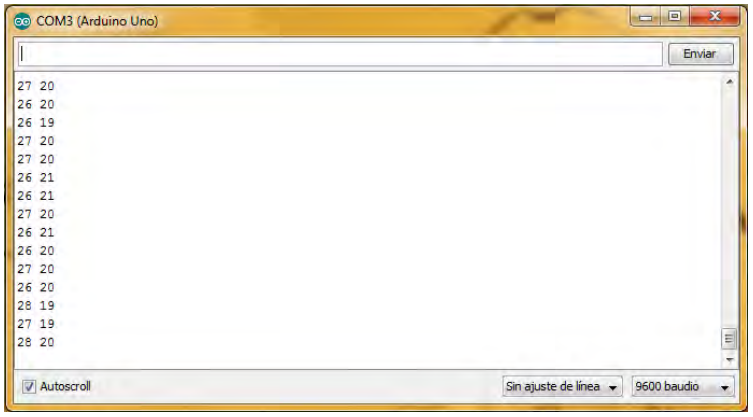

Figura 2.9: Monitor serie del entorno de programación, utilizado para enviar y recibir mensajes desde la placa

Una de las funciones mas básicas, *Serial.print()*, nos permitirá indicar a la placa que debe transmitir un mensaje a través de la comunicación serie.

A lo largo de este trabajo se utilizará el monitor serie en diversas ocasiones para obtener datos relevantes del funcionamiento de nuestros programas, como las posiciones de los servomotores o las mediciones de inclinación. Estos procedimientos se detallarán más adelante.

2.1.5.2. Librerías

Siguiendo las directrices del software en código abierto, la funcionalidad de las placas *Arduino* puede ser ampliada mediante librerías.

Las librerías son recopilaciones de funciones que proporcionan capacidades concretas a la placa para un fin específico, como por ejemplo, trabajar con hardware o manipular datos. El entorno *IDE* incluye una compilación de librerías básicas. Sin embargo, permite la importación de librerías de terceros, o incluso se puede generar la nuestra propia y añadirla a nuestro programa.

El procedimiento de adición es muy sencillo: solo hay que navegar por el interfaz hasta *Sketch* $>$ *Importar librería*. Esto añadirá una línea en la cabecera de nuestro programa con el comando *#include* seguido del nombre de la librería. También se pueden añadir librerías a nuestro programa de forma manual utilizando el mismo comando.

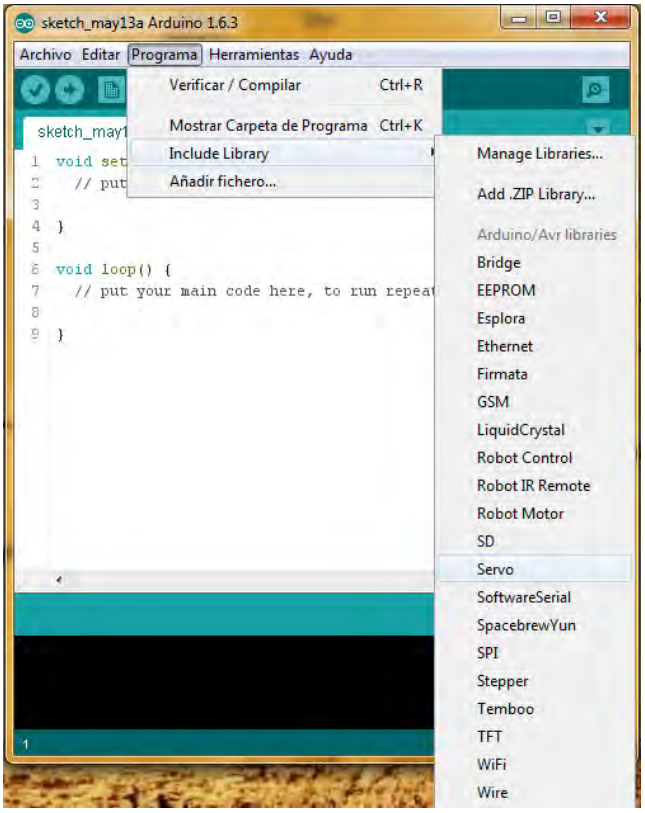

Figura 2.10: Procedimiento de adición de una librería básica de *Arduino*

2.1.5.3. Estructura básica de un programa

Los programas desarrollados en el entorno *Arduino* poseen una estructura característica que precisa ser analizada.

Todos los programas comparten una estructura común formada por dos funciones principales. Estas funciones son estrictamente imprescindibles, y deben ser declaradas en todos los programas que se ejecuten en el entorno *Arduino*, por lo que siempre se especificará qué deben hacer estas funciones.

Las funciones en cuestión se llaman *setup()* y *loop()*.

Dentro de cada función se escribirá un una parte específica del programa:

- *setup()*: en su interior se describirá el procedimiento de configuración de la placa. Este código se ejecutará sólo una vez.
- *loop()*: en su interior se incluye el cuerpo principal del código, que se ejecutará de forma repetida.

Además, el entorno *Arduino* permite declarar funciones definidas por el usuario, que comúnmente se incluyen al final del código del programa. En la figura 2.11 se puede observar la sintaxis de la declaración de una función en C, procedimiento que es idéntico en *Arduino*.

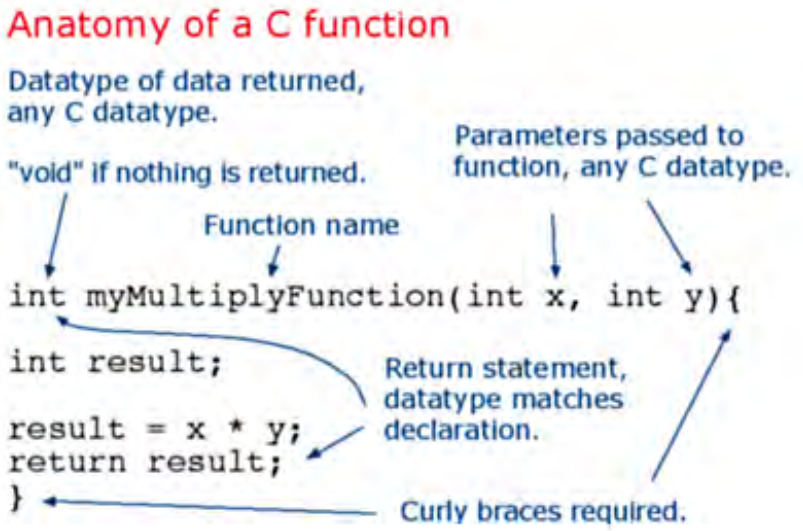

Figura 2.11: Sintaxis general de la declaración de una función en *Arduino* [7]

2.2. MATLAB

MATLAB es un lenguaje de alto nivel y un entorno interactivo comúnmente utilizado por ingenieros y científicos para el desarrollo de estudios y aplicaciones. Permite explorar y visualizar ideas, así como colaborar en diversas disciplinas incluyendo procesamiento de señales e imágenes, comunicaciones, sistemas de control y finanzas computacionales.

MATLAB se puede utilizar para cuatro finalidades concretas [15] :

- *Computación numérica*: utilizando funciones matemáticas para resolver problemas científicos e ingenieriles.
- *Análisis y visualización de datos*
- *Desarrollo de programas y algoritmos*: utilizando el lenguaje de alto nivel y las herramientas de desarrollo
- *Desarrollo de aplicaciones*: desarrollando código, ejecutables o componentes de software.

En este caso, se utilizará *MATLAB* para desarrollar un algoritmo que nos permita recopilar los datos de inclinación del acelerómetro y visualizar la orientación virtual de la antena.

2.2.1. El lenguaje de MATLAB

El lenguaje *MATLAB* proporciona soporte nativo para las operaciones de vectores y matrices que son fundamentales en la resolución de problemas científicos y de ingeniería, permitiendo un desarrollo y una ejecución rápidos.

Este lenguaje permite la escritura y desarrollo de algoritmos de forma más veloz que los lenguajes tradicionales, ya que elimina las tareas administrativas de bajo

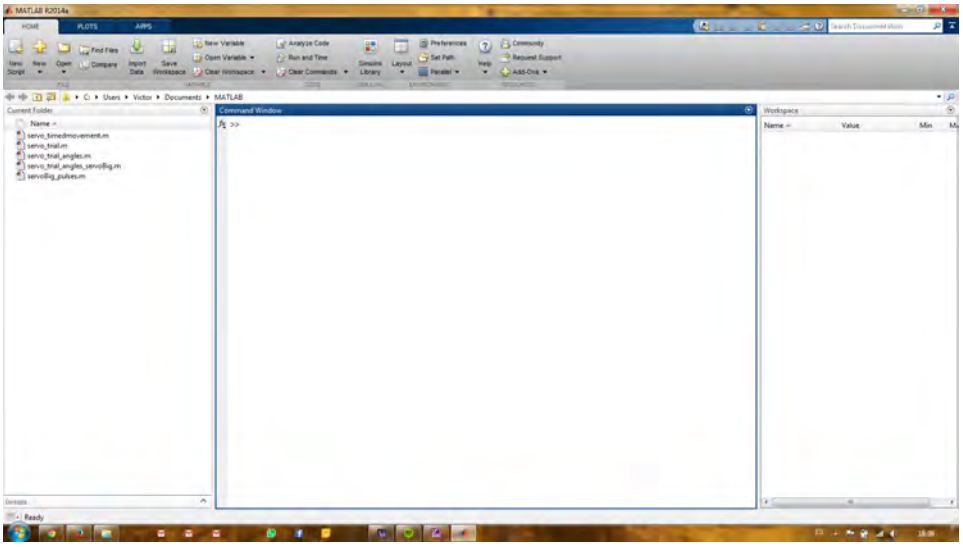

Figura 2.12: Interfaz del entorno de desarrollo *MATLAB*

nivel como la declaración de variables, la especificación de los tipos de datos o el alojamiento de las variables en memoria. Sin embargo, también incluye características de los lenguajes de programación tradicionales, como el control de flujo, el manejo de errores y la programación orientada a objetos.

Así mismo, produce resultados inmediatos mediante la ejecución de comandos. Este enfoque permite explorar rápidamente diversas opciones y visualizar los resultados intermedios para conducirnos hacia una solución óptima.

2.2.2. Herramientas de desarrollo

Las herramientas que *MATLAB* pone a disposición del usuario para el desarrollo de algoritmos se resumen en [15] :

- *Ventana de comandos*: permite la introducción interactiva de datos, ejecuta comandos y programas y muestra los resultados.

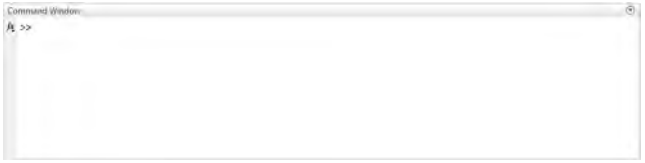

Figura 2.13: Ventana de comandos de *MATLAB*

- *Editor MATLAB*: proporciona funciones de edición y depuración, como el establecimiento de 'breakpoints' y el avance paso a paso entre líneas de código.

- *Analizador de código*: Revisa automáticamente el código en busca de problemas y recomienda modificaciones para maximizar el rendimiento y el mantenimiento.

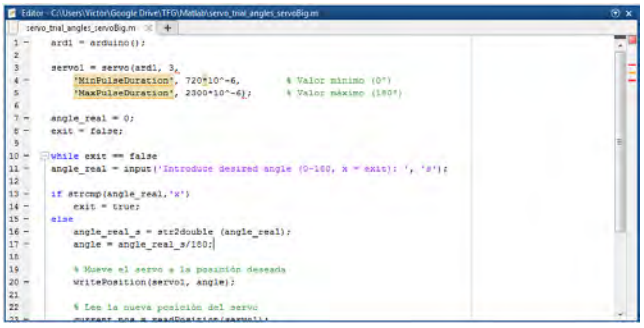

Figura 2.14: Editor de código de *MATLAB*

- *Perfilador MATLAB*: mide el rendimiento de los programas e identifica las áreas de código a modificar para mejorar el rendimiento.

2.2.3. Integración con otros lenguajes

Existe la posibilidad de integrarlas aplicaciones *MATLAB* con otras desarrolladas en otros lenguajes. Desde *MATLAB* se puede trabajar con código escrito en C, C++, Java y .NET; utilizando el motor de librerías.

2.2.3.1. Soporte de MATLAB para Arduino

Para la integración de *Arduino* en *MATLAB* se utilizará el paquete 'arduinoio' el cual ofrece el soporte necesario. Este paquete proporcionado por *MathWorks* permite, a través de un cable USB, la comunicación con *Arduino* desde la interfaz de desarrollo de *MATLAB* . El paquete está basado en un programa servidor instalado en la placa, el cual recibe los comandos transmitidos a través del puerto serie, los ejecuta y, si fuera necesario, devuelve un resultado.
El objetivo del paquete es permitir [15] :

- La programación directa sin la necesidad de otras herramientas.
- El trabajo en *MATLAB* sobre la plataforma *Arduino* manteniendo las capacidades de desarrollo y depuración.
- El desarrollo interactivo de programas para adquirir datos digitales y analógicos, así como controlar motores de corriente continua, paso a paso y servomotores.
- El acceso a dispositivos y sensores periféricos conectados mediante I^2C o SPI.

2.3. Servomotores

2.3.1. Introducción: servomotores para el control de movimiento

A continuación se lleva a cabo una descripción genérica de los servomotores y sus mecanismos. En este trabajo se utilizarán servomotores en miniatura para simular los movimientos solicitados por la placa *Arduino*.

El significado exacto de el término 'servo' en el contexto de motores y accionamientos eléctricos es difícil de acotar. En términos generales, si un motor incorpora un 'servo' en su descripción, implica que está destinado específicamente para el control en bucle cerrado. Esto nos permitirá controlar su par, su velocidad o la posición de su eje. Los primeros servomecanismos fueron desarrollados para aplicaciones militares, y pronto se hizo evidente que los motores DC (corriente continua) estándar no siempre eran los indicados para el control de posición [16] .

Ya que con este tipo de motor se busca un control preciso de su posición, se necesita que la relación par/inercia sea muy alta, de manera que el motor pueda desarrollar altos esfuerzos y después detener su movimiento de forma brusca y en la posición solicitada.

2.3.2. Tipos de servomotores

Existen múltiples tipos de motores de corriente continua, que varían su forma y tecnología en función de la aplicación. Los servomotores más populares son motores DC rotativos de imanes permanentes, PM (Permanent Magnets), que han sido adaptados a partir de motores PM DC estándar [17] .

De forma general, los servomotores PM DC se clasifican en dos tipos: con o sin escobillas. Para la aplicación que concierne este trabajo, se describirán los motores con escobillas, ya que los servomotores en miniatura se construyen comúnmente con esta configuración, exceptuando casos muy puntuales.

Servomotores PM DC con escobillas

Los motores de campo rotativo PM DC han demostrado ser fiables en aplicaciones de control donde se busca una alta eficiencia, una alto par de arranque, y una curva par-velocidad lineal [17] . Aunque comparten muchas características con los motores rotativos convencionales, los servomotores PM CD crecieron en popularidad gracias al hecho de que pueden ser controlados de forma sencilla mediante microcontroladores.

El hecho de reemplazar los bobinados generadores de campo con imanes permanentes elimina la necesidad de una excitación independiente y las pérdidas energéticas que ocurren en esos bobinados. Son motores de peso bajo, con armaduras de baja inercia que permiten acelerar más rápido que las armaduras bobinadas convencionales. Su estructura se puede observar en la figura 2.15.

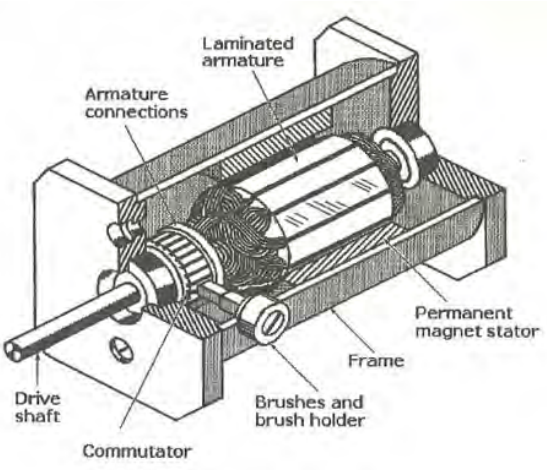

Figura 2.15: Vista en corte de un servomotor DC con rotor de imanes permanentes [17]

Las avances en circuitos integrados y microcontroladores han aumentado la fiabilidad de los controladores digitales a la vez que han permitido encapsular los motores en carcasas mas pequeñas y ligeras, y por lo tanto reduciendo el tamaño y el peso de sistemas integrados de control.

Los motores PM DC tratados en este apartado poseen una conmutación mecánica, es decir, son alimentados a través de escobillas y de un conmutador.
Mientras que todos los motores DC operan con los mismos principios, únicamente los motores PM DC poseen las curvas par-velocidad lineales mostradas en la figura 2.16, convirtiéndolos en una opción ideal para aplicaciones de lazo cerrado y de velocidad variable. Estas líneas características describen el amplio rango de rendimiento del motor. Se puede observar que tanto el par como la velocidad aumentan de forma lineal en relación al voltaje proporcionado, indicado en el diagrama con incrementos desde *V1* hasta *V5*.

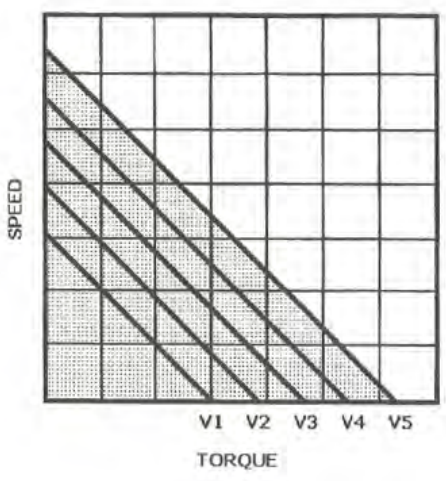

Figura 2.16: Familia de curvas par-velocidad típicas de un servomotor PM DC descritas ante diferentes voltajes de alimentación, con el voltaje aumentando de izquierda a derecha [17]

El funcionamiento de un motor de este tipo es el siguiente: los estátores de los motores PM DC con escobillas constituyen pares de polos magnéticos. Cuando el motor es alimentado, las polaridades de campo magnético generado en las bobinas y el generado por los imanes del estátor son opuestas, por lo que se atraen. Debido a esta fuerza de atracción, el rotor gira para alinearse con el estátor. En el instante en el que el rotor alcanza la posición de alineamiento, las escobillas se mueven a lo largo de los segmentos del conmutador y alimentan el siguiente bobinado.

Esta secuencia ocurre de forma reiterada siempre que el motor esté alimentado, manteniendo el rotor en un giro continuo.

2.3.3. Control de posición

Como se ha mencionado con anterioridad, los servomotores se utilizan en aplicaciones de control de posición de lazo cerrado, por lo que es apropiado analizar con detalle como se lleva a cabo este control.

En el ejemplo mostrado en la figura 2.17, el potenciómetro instalado en el eje de salida envía un voltaje de retroalimentación proporcional a la posición angular actual del eje. Mediante un voltaje de referencia, se indica cual es la posición deseada del eje.

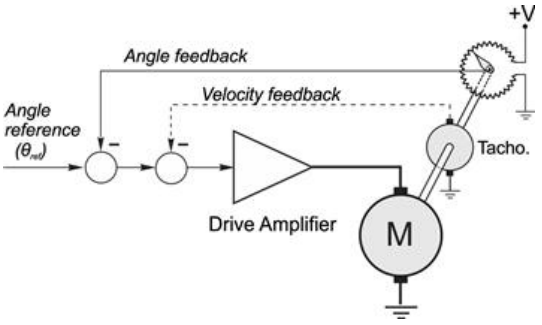

Figura 2.17: Control de posición angular en lazo cerrado utilizando un motor *DC* y una retroalimentación angular obtenida mediante un potenciómetro tipo *servo* [17]

El voltaje de la realimentación (representando el angulo real del eje) es sustraído del voltaje de referencia (que representa la posición deseada) y la señal de error de posición resultante es amplificada y usada para hacer rotar el eje de salida del motor en la dirección deseada. Cuando el eje de salida alcanza la posición objetivo, el error de posición es igual a cero, por lo que no hay voltaje aplicado en el motor y el eje de salida permanece en reposo. Cualquier intento de alejar físicamente el eje de salida de su posición objetivo se traducirá en la aparición de un error de posición y la consecuente creación de un par restaurador por parte del motor.

Con el objetivo de alcanzar una respuesta rápida y minimizar los errores de posición causados por la fricción estática, la ganancia del amplificador debe ser alta. Esto puede producir una respuesta del motor altamente oscilante, condición que es inaceptable cuando se busca un control de posición rápido y preciso [16] . Con el objetivo de mejorar la respuesta del sistema se introduce una realimentación de velocidad del eje (mostrado en la figura 2.17), llevada a cabo mediante un tacómetro.

La realimentación de velocidad no tiene efecto alguno sobre el comportamiento estático (ya que el voltaje en el tacómetro es proporcional a la velocidad del motor), pero afecta a la respuesta transitoria aumentando la amortiguación del movimiento [16] . La ganancia del amplificador se puede aumentar para proporcionar una respuesta rápida, mientras el en grado de actuación del tacómetro de puede ajustar para obtener la amortiguación deseada (ver figura 2.18).

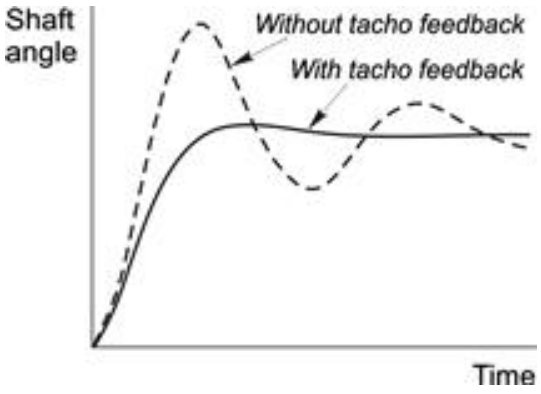

Figura 2.18: Respuesta típica ante un impulso de un sistema de lazo cerrado. Se muestra el amortiguamiento obtenido con la adición de la realimentación del tacómetro [16]

2.3.4. Servomotores utilizados

Para la realización de este proyecto se han utilizado dos servomotores en miniatura, que se muestran en la figura 2.19. El servomotor de la izquierda es un modelo *SANWA SRM-1031*, mientras que el de la derecha es el servomotor incluido en el kit de iniciación de arduino, modelo *SM-S2309S*.

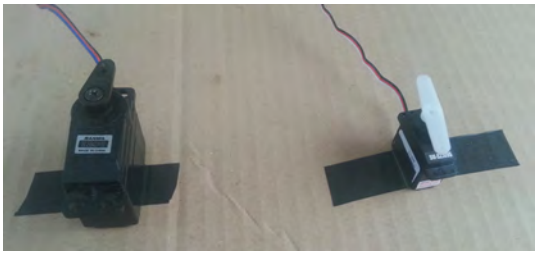

Figura 2.19: Servomotores utilizados para simular la transmisión de movimiento a la antena

Dado que se trata de motores diferentes, poseen características distintas, con discrepancias en velocidades, aceleraciones, e inercia; se procederá a la calibración de dos servomotores distintos, procedimiento que se explicará más a delante en este trabajo.

Ambos servomotores incluyen un motor PM DC con escobillas, y realizan la retroalimentación de posición a través de un potenciómetro. Están alimentados con 5V y son controlados mediante *PWM*.

En particular, el servomotor de *SANWA* es muy usual en aplicaciones de radio control. Se trata de un servo en miniatura muy robusto, que puede manejar esfuerzos de hasta 3 kgcm, recorriendo 60º en 0.2 segundos. El servomotor incluido en el kit de iniciación de *Arduino*, es más pequeño, por lo que los esfuerzos que soporta son menores (tres veces menos). Sin embargo, debido a su tamaño reducido, es el más rápido de los servomotores utilizados, recorriendo 60º en 0.12 segundos.[4]

2.3.5. El servomotor y Arduino

A continuación se incluye una breve descripción de la tecnología y la librería de *Arduino* que nos permitirá controlar el servomotor.

2.3.5.1. PWM

La modulación del ancho de pulso, *PWM* (Pulse Width Modulation en inglés), es una técnica que permite obtener resultados analógicos a través de medios digitales. El control digital se utiliza para crear una señal cuadrada con dos estados posibles, *ON* y *OFF*. Mediante este patrón *ON-OFF* se pueden simular voltajes comprendidos entre el valor de *ON* (5V en *Arduino*) y el valor de *OFF* (0V) cambiando la porción de tiempo que la señal permanece en *ON* en oposición al tiempo que la señal se encuentra en *OFF*.
La duración del tiempo *ON* se denomina 'anchura de pulso'. Para obtener diferentes valores analógicos, basta con cambiar o 'modular' esa anchura de pulso.

En la gráfica mostrada en la figura 2.20, las líneas de color verde representan un periodo de tiempo estándar, generalmente marcado por el reloj del microcontrolador. La duración del periodo modulado se calcula mediante el inverso de la frecuencia *PWM*. Por ejemplo, con la frecuencia *PWM* de *Arduino* de aproximadamente 500Hz, el espaciamiento entre las líneas de color verde será de 2 milisegundos.

La modulación se lleva a cabo mediante una llamada a la función *analogWrite()* en una escala comprendida entre 0 y 255, tal como *analogWrite(255)*, que solicita un ciclo de trabajo del 100 % (todo el ciclo *ON*). Si por ejemplo la llamada el valor de la llamada es *analogWrite(127)*, el ciclo de trabajo solicitado será del 50 %, que corresponderá a la mitad del periodo estándar.

Si se repite el patrón *ON-OFF* lo suficientemente rápido, se podrá obtener un voltaje estable comprendido entre los valores de *ON* y *OFF*. En el caso estudiado en este trabajo, este voltaje nos permitirá controlar la posición del servomotor conectado a *Arduino*.

[4] Valores para una alimentación de 4.8V

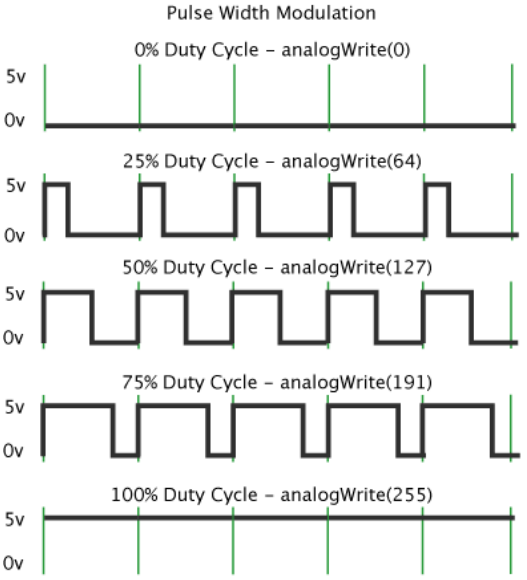

Figura 2.20: Control del ciclo de trabajo de una señal digital mediante PWM [7]

2.3.5.2. Librería Servo

La librería *Servo* proporciona a la placa *Arduino* la capacidad de controlar servomotores RC (servomotores en miniatura, de 'hobby'). Los servomotores RC estándar, como los utilizados en este trabajo, poseen un eje que puede posicionarse en valores comprendidos entre 0^o y 180^o.

Esta librería se utiliza en conjunto con los conectores *PWM* de la placa *Arduino*, de forma que se puede transmitir diferentes voltajes (comprendidos entre 0V y 5V) a través de la señal digital para controlar los servomotores. Además, permite la conexión simultánea de hasta 12 servomotores en la mayoría de placas *Arduino*.

Las funciones incluidas en esta librería son [7] :

- *attach()*: sirve para especificar en que pin está conectado el servomotor. Admite como parámetros el número del pin y las anchuras de pulso máxima y mínima del servomotor.
- *write()*: en servomotores convencionales, establece la posición deseada para el servo, solicitando dicha posición en grados (por ejemplo, 90^o para el punto medio). Para servomotores en los que el rango de movimiento (expresado en ms) no es convencional, no es suficientemente precisa.
- *writeMicroseconds()*: cumple el mismo cometido que la función anterior, pero permite transmitir el valor de posición deseado expresado en ms. Si se conoce el rango de funcionamiento del servomotor, es mucho más precisa que *write()*.
- *read()*: transmite de vuelta la posición del servomotor, expresado en grados.
- *attached()*: permite comprobar si existe un servomotor conectado a un pin determinado. Se utiliza comúnmente para detectar si el servo está correctamente conectado.

- *detach()*: sirve para desvincular el servomotor del pin al que está conectado.

La sintaxis de las mismas será descrita más adelante, cuando se lleve a cabo el análisis de los programas escritos para controlar los servomotores.

2.4. El acelerómetro

2.4.1. Principios de los sensores inerciales

La aceleración se puede obtener de la primera derivada de la velocidad o de la segunda derivada del desplazamiento. Sin embargo, el uso diferencial de las señales suele incrementar el ruido [19] .

Como consecuencia, la medida directa de la aceleración es en muchas ocasiones más fácil y conveniente. La aceleración del movimiento lineal (α), de acuerdo con la segunda ley de Newton, se expresa como la fuerza (F) actuando sobre una masa (m):

$$F = m\alpha \tag{2}$$

2.4.2. Acelerómetros

En el mercado se encuentran disponibles muchos tipos de acelerómetros de características diversas, ya que son lo suficientemente baratos y fiables como para ser usados, por ejemplo, en automóviles como sensores de impacto.

La tecnología denominada *MEMS*[5] permite producir acelerómetros muy pequeños y de alta sensibilidad.

Los acelerómetros de tipo viga son los más sensibles y comunes en el rango de medición de aceleración de cuerpos. En este tipo de acelerómetros, un extremo de una viga elástica se fija a la base del acelerómetro; y una masa, llamada la masa sísmica, se acopla al extremo opuesto, como muestra la figura 2.21.

Cuando la masa sísmica es acelerada, aparece una fuerza proporcional a la masa multiplicada por la aceleración, y la viga se flexa de forma elástica como respuesta a dicha fuerza. Para evitar la oscilación resonante tras el movimiento transitorio, se debe incluir un coeficiente de amortiguación adecuado en el interior del sistema mecánico. Un diafragma, un muelle u otro tipo de material elástico puede sustituir a la viga en este tipo de acelerómetros.

Con el objetivo de determinar la amplitud y la dirección de la aceleración en un espacio tridimensional se necesita un acelerómetro triaxial, como el mostrado en la figura 2.21.

[5] *MEMS* hace referencia al conjunto de materiales y técnicas utilizadas para la fabricación de sistemas microelectromecánicos.

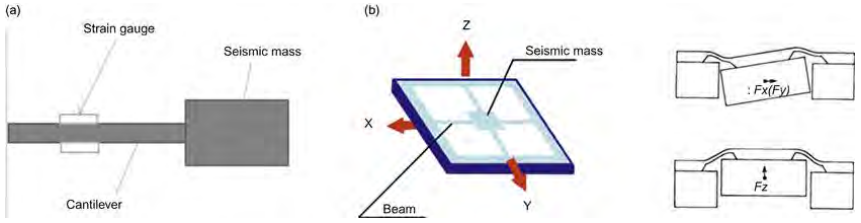

Figura 2.21: (a) Acelerómetro uniaxial y (b) acelerómetro triaxial [19]

El desplazamiento de la masa sísmica se puede detectar mediante diversos métodos. Los acelerómetros pueden estar basados en efectos capacitivos, piezorresistivos o piezoeléctricos.

- Efecto capacitivo: se utilizan generalmente para la medición de movimientos en animales o humanos, ya que poseen un tamaño reducido y son relativamente ecomómicos. En cuanto a su funcionamiento, disponen de un condensador de placas móviles. Por cada eje de medida, existen dos placas fijas y una móvil (amortiguada por muelles de polisilicona). Cuando aparece una aceleración, la placa móvil se desplaza variando la capacidad de los condensadores formados con las placas fijas. Esta variación es convertida a un voltaje de salida proporcional a la aceleración sufrida.

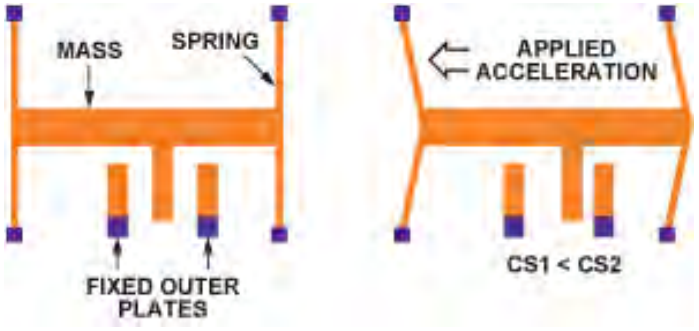

Figura 2.22: Principio de funcionamiento de una acelerómetro capacitivo

- Efecto piezorresistivo: un elemento piezorresistivo es un medidor de tensión unido una viga voladiza similar a la descrita al comienzo de esta sección. Cuando la viga se dobla en respuesta a la aceleración, la resistencia del elemento piezorresistivo cambia. De forma general, se colocan cuatro piezorresistores formando un puente de *Wheatstone*[6] que, a la vez de ofrecer la medición, elimina los errores por variaciones de temperatura y otras señales parásitas. Sirviéndose de la tecnología *MEMS*, los piezorresistores se pueden acoplar de forma sencilla a las vigas que conectan la masa sísmica con la estructura de soporte, obteniendo una alta sensibilidad.

- Efecto piezoeléctrico: los acelerómetros piezoeléctricos se usan normalmente cuando solo se quiere medir las componentes de la aceleración que varían con

[6]El puente de Wheatstone es un circuito eléctrico que permite la comparación precisa de resistencias. Al variar la resistencia de los piezorresistores, aparecerá una variación en el voltaje de salida del puente que será proporcional a la deformación sufrida [26] .

el tiempo. Éstos se caracterizan por un bajo consumo eléctrico, circuitos de detección sencillos, alta sensibilidad y una gran estabilidad ante variaciones de temperatura. Cuando se produce una aceleración, aparece un voltaje de polarización en el elemento piezoeléctrico que es proporcional a su deformación. La polaridad de este voltaje depende de la estructura molecular del elemento utilizado. La figura 2.23 presenta un ejemplo del principio de funcionamiento de un acelerómetro piezoeléctrico.

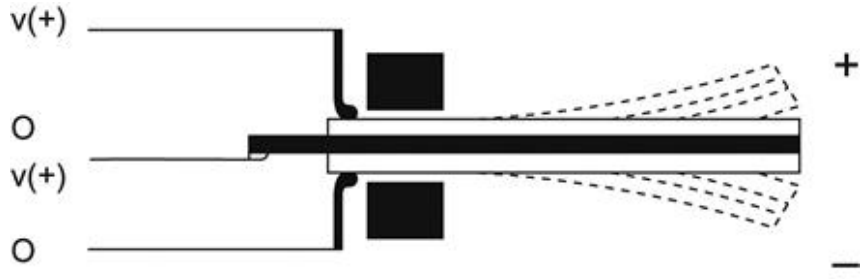

Figura 2.23: Principio de funcionamiento de un acelerómetro piezoeléctrico [19]

El voltaje del terminal del sensor piezoeléctrico es producido por la carga eléctrica que aparece debido a la flexión del elemento piezoeléctrico y la consecuente variación de su capacidad.

En conclusión, las señales de aceleración se utilizan en un amplio rango de medidas y aplicaciones, incluyendo medidas de equilibrio, transferencias de reposo a movimiento, clasificación de movimientos y rastreo de actividades físicas.

Medición de la inclinación

Este proyecto, de forma concreta, se centrará en las mediciones de inclinación del acelerómetro. De esta forma, al inclinar la mano, el acelerómetro detectará esa variación y la transmitirá a los servomotores.

Para dicha función, se utilizará un acelerómetro triaxial que se encargará de medir el campo gravitatorio estático, compuesto por una aceleración de $1g = 9{,}8\ \mathrm{m/s^2}$. Al variar la inclinación (a lo largo del eje de medida) se cambiará el valor del vector que define la aceleración, como se muestra en la figura 2.24.

Ya que la gravedad es estática, se debe realizar la medición en corriente continua (DC). Para alcanzar una medida adecuada, el sensor debe proporcionar una alta repetitividad de las medidas y una amplia resolución, que permita transmitir las pequeñas variaciones del campo gravitatorio que actúa sobre el acelerómetro al proceder a la inclinación [20] .

2.4.3. Protocolo SPI

A lo largo del desarrollo de este trabajo, la comunicación con el acelerómetro utilizado se realizará mediante el protocolo SPI.

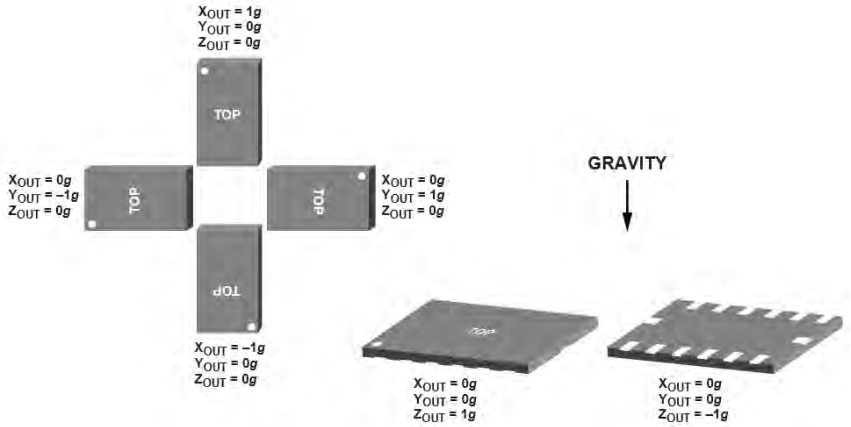

Figura 2.24: Medidas de salida del acelerómetro en reposo

2.4.3.1. Introducción a los buses de comunicación

En este contexto, un bus es un canal de comunicación en el que existen uno o varios transmisores y múltiples receptores. Todos los receptores pueden ver la información que se transmite a través del canal de comunicación. Cada receptor descodifica todos los mensajes transmitidos y utiliza una dirección incluida en el mensaje para determinar si es el objetivo de la transmisión. El receptor objetivo del mensaje contesta a través del mismo canal de comunicación [21] .

La figura 2.25 ilustra dos ejemplos de buses de comunicación. En la parte (a) de la figura 2.25 se muestra como una conversación normal entre amigos esta basada esencialmente en una comunicación bus; en la que el transmisor utiliza el nombre de la persona como dirección al enviar el mensaje a través del canal de comunicación, que es aire en este caso. La persona que posee ese nombre responde al mensaje.

La parte (b) muestra como las direcciones de Internet se utilizan en una red Ethernet para habilitar la comunicación entre ordenadores. Una red Ethernet es un bus, en el que todos los ordenadores monitorizan el tráfico de información, y solo responden a aquellos paquetes de datos que incluyen en su encabezamiento la dirección asociada al ordenador receptor.

2.4.3.2. Comunicación serie

La comunicación serie es un proceso en el que se transmite una información determinada de forma secuencial. La transmisión se hace a través de un bus similar al descrito en el apartado anterior.

En oposición a la comunicación en paralelo, la transmisión se realiza bit a bit y por un único canal de comunicación. La comunicación serie se utiliza en transmisiones de larga distancia y en la mayoría de redes de ordenadores donde el coste del cableado y las dificultades de sincronización hacen imposible la comunicación en paralelo.

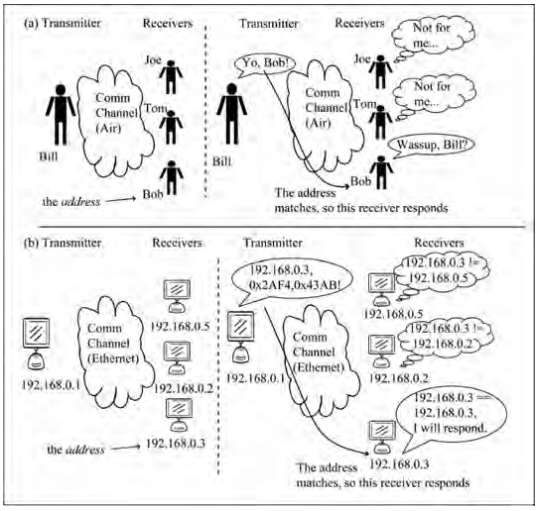

Figura 2.25: Dos ejemplos de comunicación bus [21]

Sin embargo, la comunicación serie ofrece una mayor integridad de la señal transmitida, y los últimos avances han permitido un gran aumento en la velocidad de transmisión. Estos factores han propiciado la sustitución de las comunicaciones en paralelo por comunicaciones serie, incluso en distancias cortas.

2.4.3.3. Serial Peripheral Interface (SPI)

La interfaz serie de periféricos (SPI, pronunciado como 'spy' en inglés) es muy similar a la comunicación serie estándar. Proporciona una comunicación dúplex [7] a la vez que una conexión síncrona entre varios dispositivos.

La interfaz SPI presenta las siguientes características [12] :

- Puede alcanzar altas velocidades de transmisión. Para componer el bus de comunicación se suele emplear un único dispositivo maestro y uno o varios dispositivos esclavos.
- No existe un límite para la señal del reloj establecido por el estándar SPI. Por lo tanto, está únicamente limitada por la máxima velocidad del reloj del hardware.
- La señal del reloj es proporcionada por el dispositivo maestro y se comparte entre todos los dispositivos conectados al bus, por lo que se elimina la necesidad de tener que proporcionar esta señal a los esclavos de forma individual.
- El estándar SPI define varios tipos de conexión del bus: una configuración con tres cables, que utiliza una línea de transmisión bidireccional con un método semi-dúplex [8]; la configuración más común con cuatro cables; y la configuración que utiliza cinco cables, que añade una línea de transmisión que permite al dispositivo esclavo informar al maestro de que está preparado para realizar la transmisión.

[7] Un sistema dúplex es aquel que permite la comunicación bidireccional, es decir, el envío y recepción de mensajes de forma simultánea.

[8] En una comunicación semi-dúplex, los datos se transmiten en una u otra dirección, no pudiendo ocurrir de forma simultánea.

Con el objetivo de facilitar la lectura de este capítulo, se incluye la siguiente tabla como referencia de las abreviaciones utilizadas:

Abreviación	Definición
SCK (serial clock)	Señal de reloj generada por el dispositivo maestro del bus SPI.
SS (slave select)	Señal de valor lógico alto o bajo utilizada para seleccionar uno o varios dispositivos esclavos.
MOSI (master out slave in)	Línea de transmisión compartida por todos los dispositivos pertenecientes al bus. Constituye la línea de datos de salida del maestro y de entrada de los dispositivos esclavos.
MISO (master in slave out)	Línea de transmisión compartida por todos los dispositivos pertenecientes al bus. Constituye la línea de datos de entrada del maestro y de salida de los dispositivos esclavos.

En *Arduino*, el valor lógico bajo[9] corresponde a 0V y el valor lógico alto a 5V en los pines digitales.

Conectando dos dispositivos

El punto de partida para configurar cualquier bus SPI es la librería del dispositivo maestro. Los dispositivos esclavos se configuran a través de la manipulación directa de sus registros, ya que la librería SPI no incluye funciones para dichos dispositivos. A continuación se enumeran las funciones principales de la librería SPI:

- *SPI.begin()*: inicia la comunicación SPI y establece los pines de entrada y salida del bus.
- *SPI.end()*: finaliza la comunicación SPI, pero no modifica la configuración de los pines.
- *SPI.setBitOrder()*: establece si la comunicación es LSBFIRST (primero se transmite el bit menos significativo) o MSBFIRST (primero se transmite el bit más significativo). El dispositivo maestro y los esclavos deben tener el mismo modo de comunicación. A excepción de algunos dispositivos concretos, que requieren un orden de transmisión específico, por lo general esta configuración es arbitraria.
- *SPI.setClockDivider()*: Arduino es capaz de manejar un bus SPI a diferentes velocidades a través de un divisor de la señal del reloj. Esta función es útil cuando los dispositivos conectados al bus no pueden trabajar a la velocidad máxima de Arduino.

  Es importante resaltar que los dispositivos esclavos deben ser capaces de trabajar a la velocidad de reloj marcada por el maestro. Si son muy lentos, podrían fallar en la lectura del bus.

[9] Los valores lógicos de un pin digital pueden ser alto (1 digital) o bajo (0 digital).

- *SPI.setDataMode()*: determina la configuración del reloj y el momento de lectura del bus. Las transmisiones del bus son leídas por los dispositivos en un momento determinado que depende del ciclo del reloj.

  Esta configuración es similar al baud rate de las comunicaciones serie, y todos los dispositivos deben configurarse del mismo modo con el fin de crear un transmisión de datos plausible y eficaz.

  El cuadro 2.1 muestra las distintas configuraciones y el momento de lectura del bus en relación a a polaridad del reloj. Si se utiliza la configuración mode0, el reloj estará en reposo en su valor lógico bajo, y el bus será leído en el flanco de subida del reloj. El mode1 comparte la misma polaridad de reloj que el mode0, pero la lectura del bus ocurre en el flanco de bajada. El mode2 y el mode3 imitan el funcionamiento de las otras dos configuraciones, pero el estado de reposo del reloj en este caso es su valor lógico alto.

Cuadro 2.1: Modos de transmisión de datos en SPI

Comando	Modo	Lectura del bus	Valor lógico del reloj en reposo
SPI_MODE0	0	Flanco de subida	Bajo
SPI_MODE1	1	Flanco de bajada	Bajo
SPI_MODE2	2	Flanco de subida	Alto
SPI_MODE3	3	Flanco de bajada	Alto

  Como se ha explicado con anterioridad, en el mode2 y el mode3 el estado de reposo del reloj es un valor lógico alto (en *Arduino*, 5V). por lo tanto, dado que el reloj pasa la mayor parte de su tiempo en reposo, incluso durante las transmisiones, estas configuraciones no son recomendables en aplicaciones dónde se busque un mínimo consumo de energía.

- *SPI.transfer()*: transfiere 1 byte de datos a través del bus SPI. Esta función devuelve los datos recibidos desde el dispositivo esclavo activo en ese momento. Se trata de una transmisión dúplex: el bit enviado por el maestro se transfiere hacia el esclavo a la vez que el esclavo envía 1 bit hacia el maestro; por lo tanto, se lleva acabo un intercambio simultáneo de información.

Configurando el dispositivo maestro

Debido a los diferentes roles que desempeñan los dispositivos en el bus SPI, la configuración de los pines será diferente en función de la tarea que desempeñen, como se muestra en el cuadro 2.2:

Cuadro 2.2: Configuración de los pines de los dispositivos en un bus SPI

Maestro			Esclavo	
MOSI	Salida	$\rightarrow$	MOSI	Entrada
MISO	Entrada	$\leftarrow$	MISO	Salida
SCK	Salida	$\rightarrow$	SCK	Entrada
SS	Salida	$\rightarrow$	SS	Entrada

Para el dispositivo maestro, el pin SS debe estar configurado como salida. Si este pin está configurado como entrada y toma su valor lógico bajo, el maestro perdería su configuración y comenzaría a comportarse como un dispositivo esclavo.

El siguiente fragmento de código incluye un ejemplo de la configuración de un dispositivo maestro SPI en *Arduino* [21] :

```
#include <SPI.h> // Incluye la librería SPI para trabajar con el dispositivo
    maestro
byte dataToSend;
byte dataToReceive;

void setup() {
  pinMode(10,OUTPUT); // Configura el pin SS (slave select) del maestro como salida
    . El dispositivo maestro en este caso es Arduino
  digitalWrite(10, HIGH); // Fija el pin SS en su valor lógico alto para poder
    iniciar la comunicación
  SPI.begin(); // Inicia el bus SPI
  Serial.begin(115200); // Comienza una comunicación serie con el ordenador
  delay(500); // Permite la inicialización de los dispositivos conectados
}// Finaliza la función de configuración

void loop() {
  while (Serial.available() > 0) {
    dataToSend = Serial.read(); // Lee un valor de la comunicación serie (
    introducido por el usuario)
    transferSPI(dataToSend); // Envía ese byte
  }
  delay(1000);
}// Finaliza el bucle de transmisión

// La siguiente función realiza el proceso de transmisión
byte transferSPI(byte dataToSend) {
  digitalWrite(10, LOW); // Fija el pin SS en su valor lógico bajo para señalar el
    comienzo de la transmisión de un paquete SPI
  delay(1); // El esclavo tarda un tiempo en responder al cambio de valor del SS
  dataToReceive = SPI.transfer(dataToSend); // Inicia la comunicación dúplex
  digitalWrite(10, HIGH); // Fija el pin SS en su valor lógico alto para finalizar
    la comunicación con el esclavo
  Serial.write(dataToSend); // Se muestran los datos enviados
  Serial.println();
  Serial.write(dataToReceive); // Se muestran los datos recibidos
  Serial.println();
}// Finaliza la comunicación SPI
```

Actuando sobre los registros

Como se ha indicado con anterioridad, no existe una librería SPI que permita configurar los dispositivos esclavos, por lo que se debe actuar de forma manual sobre éstos.

A través de una serie de ejemplos de código se mostrará como se lleva a cabo esa configuración y se explicará una función modelo modelo que servirá para modificar los registros.

Debido a la estructura de los programas de Arduino, mencionada con anterioridad, se configurará el dispositivo esclavo en el interior de la función *setup()*.

A continuación se incluye un extracto del código utilizado para configurar el acelerómetro utilizado en este trabajo:

```
#include <SPI.h> // Incluye la librería SPI para trabajar con el dispositivo
    maestro

int CS=10; //Asigna el pin SS al pin 10. En ocasiones Slave Select se expresa como
    Chip Select

#define POWER_CTL 0x2D  // Registro de control de energía del acelerómetro
#define DATA_FORMAT 0x31  // Registro del formato de los datos del acelerómetro

void setup(){
  SPI.begin(); // Inicia la comunicación SPI
  SPI.setDataMode(SPI_MODE3); // Configura el bus SPI para comunicarse con el
    acelerómetro. En este caso se escoge el mode3
  pinMode(CS, OUTPUT); // De nuevo, el dispositivo maestro es el Arduino, por lo
    que se configura el pin SS como salida
  digitalWrite(CS, HIGH); // Fija el pin SS en su valor lógico alto para poder
    iniciar la comunicación

  writeRegister(DATA_FORMAT, 0x00); // Escribiendo el valor especificado (0x00) en
    el registro DATA_FORMAT del acelerómetro, se configura para que tome medidas
    en un rango de +/- 2g (la dirección de estos registros suele aparecer en el
    datasheet del dispositivo)

  writeRegister(POWER_CTL, 0x08);  // Inicia el modo 'medición' del acelerómetro
    escribiendo el valor necesario en el registro de control de energía, POWER_CTL
```

Como se puede observar en el fragmento anterior, se ha utilizado una función personalizada denominada *writeRegister* para acceder a los registros del dispositivo esclavo.

Dicha función es muy similar a la utilizada en el apartado anterior, *transferSPI(byte dataToSend)*. Su código se puede revisar a continuación:

```
void writeRegister(char registerAddress, char value){
  digitalWrite(CS, LOW); // Fija el pin SS en su valor lógico bajo para señalar el
    comienzo de la transmisión de un paquete SPI
  SPI.transfer(registerAddress); // Transfiere la dirección del registro a través
    del bus SPI
  SPI.transfer(value); // Transfiere el valor deseado a través del bus SPI
  digitalWrite(CS, HIGH); // Fija el pin SS en su valor lógico alto para finalizar
    la comunicación con el esclavo
}
```

2.4.4. ADXL345

El acelerómetro utilizado para la consecución de este proyecto es el modelo ADXL345 de *SparkFun*, el cuál se muestra en la figura 2.26.

Figura 2.26: ADXL345, acelerómetro triaxial utilizado [24]

2.4.4.1. Introducción

Se trata de un acelerómetro triaxial de alta resolución (13 bits)[10], constituido por un perfil pequeño, delgado, de bajo consumo y con medidas comprendidas en un rango de hasta 16g (la resolución cambia en función del rango escogido). Los valores digitales de salida son formateados en 16 bit de complemento a dos[11], y se pueden obtener a partir de un bus de configuración SPI o con una interfaz digital del protocolo I^2C.

Este acelerómetro es adecuado para aplicaciones dirigidas a dispositivos móviles, ya que es capaz de medir la gravedad estática en aplicaciones de medición de inclinación, así como la aceleración dinámica que aparece como resultado de movimientos o impactos. Su alta resolución (4mg/LSB)[12] permite medidas de inclinación menores de 1.0º.

Así mismo incluye algunas funciones especiales. Es capaz de detectar actividad o inactividad mediante la presencia o ausencia de movimiento. Cuando el valor de la aceleración supera en cualquiera de los ejes un valor preestablecido por el usuario, el sensor envía una señal comunicando la aparición de movimiento, lo que se denomina como la adición de un umbral de inactividad a la detección de movimiento. También es capaz de interpretar episodios de caída libre.

Además, este dispositivo incluye un buffer *FIFO* [13] que puede ser utilizado para almacenar datos y minimizar el trabajo del microcontrolador huésped.

[10]Los valores medidos por el acelerómetro podrán alcanzar hasta 13 dígitos expresados en código binario. Por ello, para las medidas se utilizan dos bytes de información (16 bits).

[11]Sistema de representación binario que nos permite escribir números negativos, siendo el bit de signo el MSB (most significant bit).

[12]Los puertos digitales tienen una resolución de 1024 bits, o 1024 LSB. Por lo tanto, por cada LSB, el acelerómetro es capaz de detectar y transmitir la milésima parte de una aceleración de intensidad 4g.

[13]Buffer del tipo 'first in, first out'.

2.4.4.2. Principio de funcionamiento

El sensor esta constituido por una estructura mecanizada mediante la tecnología *MEMS* sobre una oblea de silicio, y utiliza el efecto capacitivo para realizar las mediciones.

Como se detalló en la sección 2.4.2, se utilizan condensadores diferenciales, los cuáles consisten en dos placas: una fijada a la estructura y la otra acoplada a la masa móvil (masa sísmica). La aceleración flexa la viga que sujeta la masa sísmica y desequilibra el condensador diferencial, lo que produce la aparición de una señal de salida del sensor que es proporcional a la aceleración sufrida.

En la figura 2.27 se puede observar el diagrama de bloques funcional del sensor descrito.

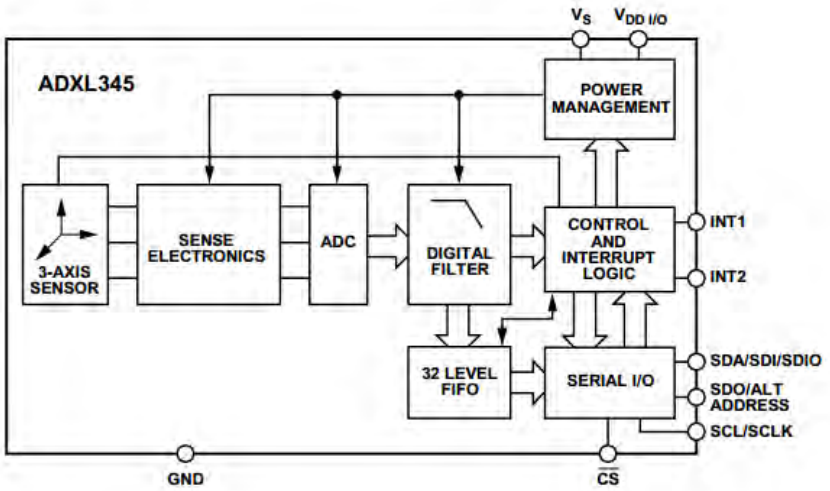

Figura 2.27: Diagrama de bloques funcional del ADXL345

Como se puede observar en el diagrama de bloques, las salidas de los condensadores diferenciales se transforman a valores digitales mediante un conversor ADC[14], y la señal se filtra para eliminar el ruido. Tras el filtro, se encuentran los circuitos lógicos de los interruptores que informan de la aparición de movimiento (INT1 e INT2) y el buffer *FIFO*.

Por último, se encuentra el módulo de comunicación serie, que controla los intercambios de información.

En el diagrama se advierte que el pin SS (slave select), que en el ADXL345 se expresa como CS (chip select), funciona con lógica inversa. Ésto explica por qué al comenzar la transmisión del paquete SPI, el pin SS se fija en su valor lógico bajo, indicador de que el proceso de transmisión va a comenzar.

[14]Un conversor ADC convierte una señal analógica en una señal digital.

Debido a que este acelerómetro también se puede utilizar a través de una interfaz I^2C, la nomenclatura del diagrama de bloques difiere de la utilizada para el bus SPI en la sección 2.4.3.3, por lo que en la siguiente tabla se describen las equivalencias entre ambas nomenclaturas:

Bus SPI		ADXL345
MOSI	$\rightarrow$	SDO
MISO	$\rightarrow$	SDA
SCK	$\rightarrow$	SCL
SS	$\rightarrow$	CS

La función principal que se utilizará en este trabajo será la de medición de inclinación, ya que es la aplicación que más se adapta a nuestro objetivo.

Sin embargo, las funciones adicionales de detección de caída libre y de umbral de inactividad se podrán utilizar en trabajos futuros, buscando exprimir el potencial de este dispositivo.

Capítulo 3

Desarrollo de métodos experimentales

En este capítulo se procede a la ilustración, explicación y desarrollo de los programas y experimentos producidos durante el desarrollo del trabajo.

Se explicará en que consiste cada método de control planteado y, para cada uno de ellos se describirán: el planteamiento teórico, los materiales utilizados, el desarrollo del método y los resultados y conclusiones alcanzados.

3.1. Método 1º. Sensor capacitivo

3.1.1. Planteamiento del método

Como primera aproximación al control remoto de gestos, procederemos a la construcción de un sensor capacitivo para valorar sensibilidad y la factibilidad de su utilización en la consecución de nuestro objetivo.

El sensor construido está basado en la configuración que se expone en instructables.com, en el artículo titulado 'DIY 3D Controller' [27] .

En resumen, nuestro objetivo será obtener la posición de nuestra mano en un entorno de tres dimensiones, y a continuación, traducir esta información de forma que proporcione instrucciones de movimiento a los servomotores.

3.1.2. Principio de funcionamiento

El sensor funcionará a través de varios circuitos RC simples, y cada uno de los circuitos detectará la distancia a la que se encuentra la mano en cada una de las dimensiones.

El la figura 3.1 se puede observar un esquema del principio de funcionamiento de éste método.

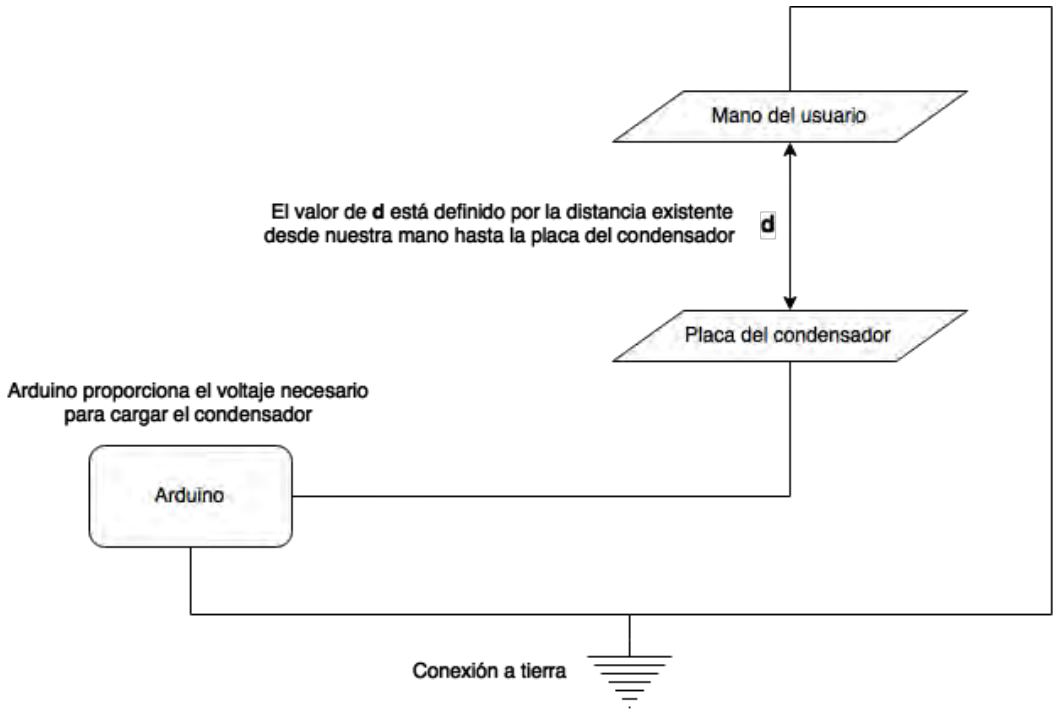

Figura 3.1: Esquema del principio de funcionamiento del sensor basado en un condensador de placas variable

3.1.2.1. Circuito RC

Se trata de un circuito eléctrico compuesto por resistencias y condensadores alimentados por una fuente de tensión o de corriente. El circuito *RC* más simple está compuesto por una sola resistencia y un solo condensador.

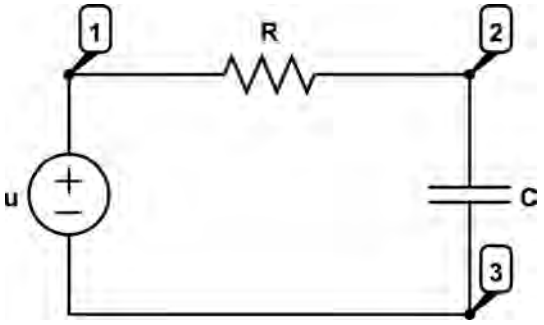

Figura 3.2: Circuito RC simple

Cuando este circuito se conecta a una fuente de alimentación de corriente alterna, éste funciona como un filtro *RC*, que bloqueará ciertas frecuencias en función de los valores de *R* y de *C*.

Sin embargo, para conseguir la medición que nosotros deseamos, conectaremos el circuito a una fuente de voltaje de corriente continua, función que cumplirá nuestra placa *Arduino*. Conectado de esta forma, un circuito *RC* cargará el condensador hasta alcanzar su capacidad máxima. La intensidad irá disminuyendo de forma ex-

ponencial conforme el condensador adquiere carga, hasta que se alcance su carga máxima, momento en el cuál se comportará como un circuito abierto.

Como se detallará más adelante, interesa establecer una relación entre la capacidad del condensador y su tiempo de carga, ya que utilizaremos la variación en su tiempo de carga para transformarla en el movimiento de un servomotor. Para ello, debemos recurrir a la expresión diferencial de la intensidad que recorre el circuito al que está conectado el condensador:

$$i(t) = \frac{dq}{dt} \tag{1}$$

siendo:

- $i(t)$: valor de la intensidad que circula por la sección del conductor
- q: carga que atraviesa la sección en una unidad de tiempo

Además, el valor de la capacidad del condensador se define como:

$$C = \frac{q}{u} \tag{2}$$

siendo:

- q: valor de la carga almacenada en el condensador
- u: diferencia de potencial aplicada entre las placas del condensador

En un circuito como el de la figura 3.2:

- El potencial que cae en la resistencia R se puede expresar mediante la ley de Ohm como $V_{12} = iR$.
- El potencial entra las placas del condensador se puede expresar como $V_{23} = q/C$, como indica la ecuación (2).
- El potencial que proporciona la fuente de alimentación se define como u_0.

Por lo tanto, el voltaje u_0 se puede expresar como:

$$u_0 = iR + q/C \tag{3}$$

Sustituyendo la ecuación (1) en la ecuación anterior, e integrando:

$$R\frac{dq}{dt} = u_0 - \frac{q}{C} \tag{4}$$

$$\int_0^q \frac{dq}{Cu - q} = \frac{1}{RC}\int_0^t dt \tag{5}$$

$$q = Cu_0(1 - e^{(\frac{-t}{RC})}) \tag{6}$$

Derivando ahora con respecto al tiempo, obtenemos la expresión que define la intensidad en el circuito RC:

$$i(t) = \frac{dq}{dt} = \frac{u_0}{R} e^{(\frac{-t}{RC})} \tag{7}$$

Que también se puede expresar como:

$$u(t) = u_0(1 - e^{(\frac{-t}{RC})}) \tag{8}$$

El comportamiento descrito por las ecuaciones (7) y (8) se puede observar en la figura 3.3.

El valor de RC, que aparece como denominador de t, se denomina constante de tiempo y se denomina con la letra τ. Expresa el tiempo necesario para que la intensidad que discurre por el circuito durante la carga del condensador decrezca hasta $\frac{1}{e}$.

El tiempo de carga de un condensador es teóricamente infinito, ya que el voltaje entre sus bornes (en corriente continua) aumenta de forma exponencial hasta alcanzar u_0. Sin embargo, el tiempo de carga se puede aproximar al tiempo que tarda el condensador en alcanzar el 99 % de su carga máxima, que se define como:

$$t_c = 5\tau = 5RC \tag{9}$$

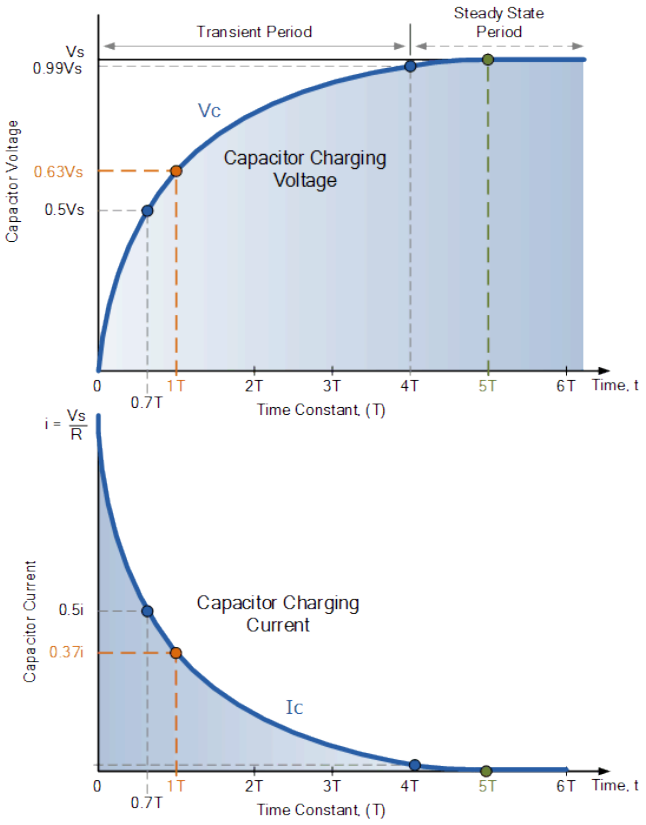

Figura 3.3: Voltaje y corriente durante la carga de un condensador [33]

3.1.2.2. Funcionamiento

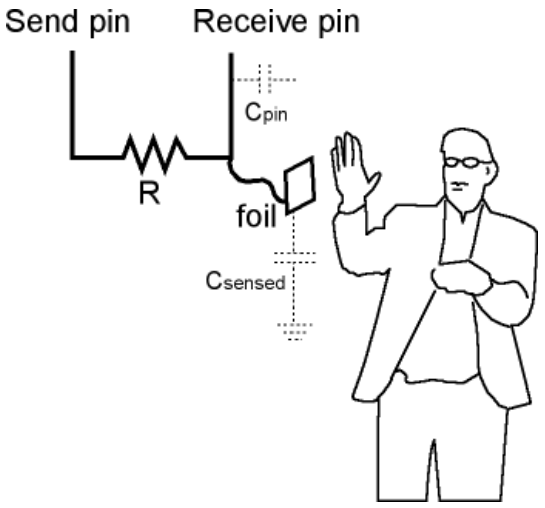

Figura 3.4: Esquema del funcionamiento de la detección capacitiva a través de la utilización de dos pines de *Arduino*. En el caso tratado en este trabajo, sólo se utiliza un pin, a pesar de que el principio de funcionamiento es exactamente el mismo [7]

El fundamento del diseño desarrollado para este proyecto radica en la creación de un condensador de placas variables, en el que nuestra mano actúa como una de las placas. Por lo tanto, seremos capaces de variar la capacidad del condensador a través del movimiento de nuestra mano, como se describirá en detalle más adelante.

En la figura 3.5 se puede observar un esquema del circuito construido.

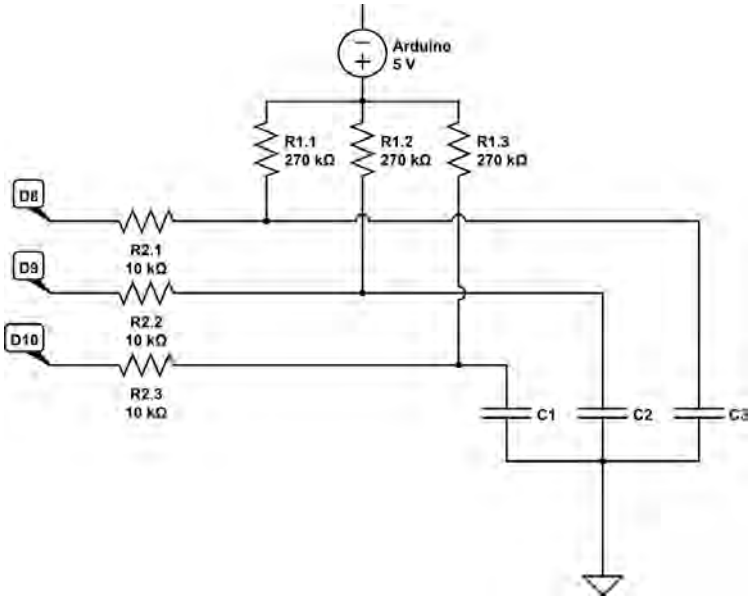

Figura 3.5: Circuito que constituye el sensor capacitivo

- D8, D9 y D10 denotan los pines digitales de *Arduino*, a los que se conectarán las tres ramas del sensor.
- Las resistencias de $10k\Omega$ limitan la corriente de entrada a los pines de *Arduino*, y aseguran que mientras se está cargando el condensador, toda la corriente circula entre la fuente de la alimentación y la toma de tierra (pasando por la resistencia pull-up y por el condensador).
- C1, C2, y C3 son los condensadores formados con las placas de papel de aluminio y la mano. Variarán su capacidad en función de la distancia de la mano a la placa.
- Las resistencias de $270k\Omega$ actúan como pull-up. Una resistencia pull-up se coloca entre el conductor de una señal y una fuente de corriente positiva para asegurarse de que se obtiene un valor lógico alto válido en caso de que la señal externa se desconecte o tenga una alta impedancia, como muestra la figura 3.6.

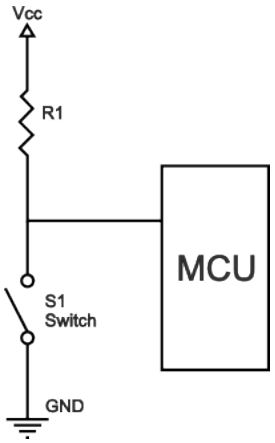

Figura 3.6: Resistencia pull-up conectada a la entrada de un microcontrolador (Microcontroller Unit). [34]

En el caso estudiado en este proyecto, el 'Switch' de la figura 3.6 es sustituido por el condensador de placas variables. Como se está trabajando en corriente continua, cuando este condensador se carga, se comporta como un circuito abierto, de manera que la resistencia pull-up forzará el pin de *Arduino* a alcanzar su valor lógico alto, es decir, 5V. Por ello las resistencias están conectadas a la fuente de 5V de *Arduino*.

Durante el tiempo en el que el condensador se está cargando, existirá una tensión de 5V entre sus bornes, que irá disminuyendo de forma exponencial como describe la ecuación (8). El tiempo de carga del condensador estará definido por los valores de la capacidad del condensador *C* (variable) y de la resistencia *R* (en este caso, $270k\Omega$), como expresa la ecuación (9). Debido a que la resistencia tiene un valor constante, cualquier cambio en la capacidad del condensador creado puede ser medida. La distancia a nuestra mano (que actúa como una placa del condensador conectada a tierra) será la variable principal en la variación de la capacidad.

De esta forma, usando los puertos digitales de *Arduino*, el funcionamiento de nuestro sensor será el siguiente:

1. Primero, configuramos el pin en modo 'output' (salida).
2. Ahora forzamos el pin a su valor lógico bajo (0 digital), lo que conecta el pin a tierra. Esto se traduce en que ambos lados del condensador están conectados a tierra, y por tanto, éste se descarga.
3. Una vez descargado, configuramos el pin en modo input (entrada).
4. Por último, medimos cuanto tiempo tarda el pin en alcanzar su valor lógico alto.

3.1.3. Conexiones

3.1.3.1. Conexión del sensor

Para conectar el sensor a *Arduino* debemos utilizar cable apantallado. En caso contrario, si el cable fuera suficientemente largo, se comportaría como la mayor parte del condensador, y no se podrían medir las perturbaciones generadas por el movimiento de la mano.

Como describe la figura 3.5, se conectará el sensor a los puertos digitales 8, 9 y 10 de la placa *Arduino*, y los 5V representados en el circuito se obtendrán de la misma placa, ya que dispone de dos salidas de corriente de 3V y 5V respectivamente. A través de las resistencias de $270\ \mathrm{k}\Omega$ se proporciona el voltaje que cargará los condensadores. Cuánto mayor sea su valor, mayor será el tiempo necesario para completar la carga.

El sensor está constituido por la esquina de un cubo de 21.6cm de lado. Cada superficie interior de dicha esquina está recubierta de papel de aluminio, que actúa como la placa fija del condensador. Para conectar las placas a *Arduino*, se han construido unos conectores constituidos por el cable apantallado, las resistencias y unas pinzas de tipo cocodrilo.

El aspecto de los conectores se muestra a continuación:

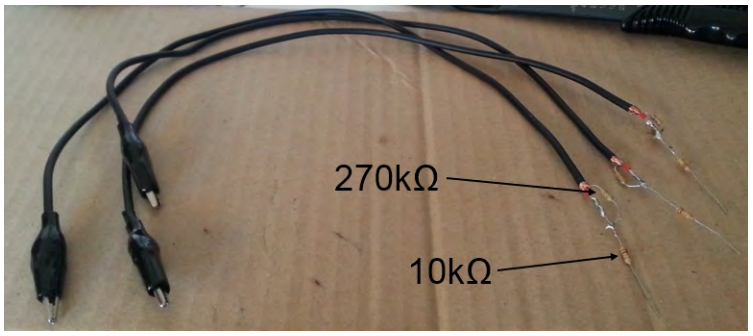

Figura 3.7: Aspecto de los conectores separados y sin sellar. Se observan con claridad las resistencias soldadas a la pantalla del cable y al interior del mismo

Figura 3.8: Conectores soldados a los pines de entrada para *Arduino*

Para alimentar el apantallado del cable utilizaremos los mismos 5V que hemos utilizado para cargar los condensadores. El circuito, ya conectado al sensor y preparado para medir tiene el aspecto siguiente:

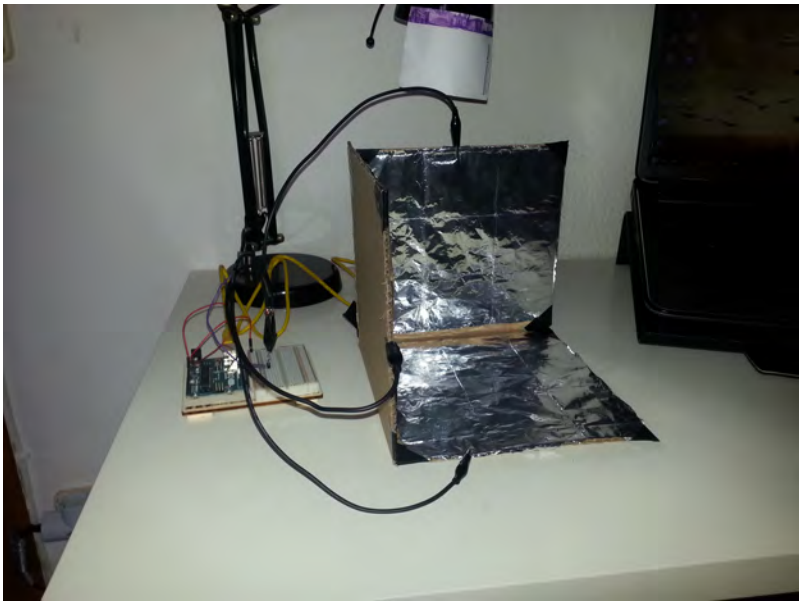

Figura 3.9: Sensor conectado a *Arduino* y preparado para medir

3.1.3.2. Conexión de los servomotores

Los servomotores se alimentan a través del puerto de 5V que ofrece la placa *Arduino*. El cable de control deberá ir conectado a una entrada digital de la placa que posea la capacidad PWM, para poder controlar el motor correctamente.

Además, para evitar los posibles picos de tensión al invertir la dirección de la corriente en el motor durante su funcionamiento, añadiremos un condensador de 100μF entre los bornes de alimentación del servomotor.

El esquema de la configuración descrita se puede observar en la figura 3.10.

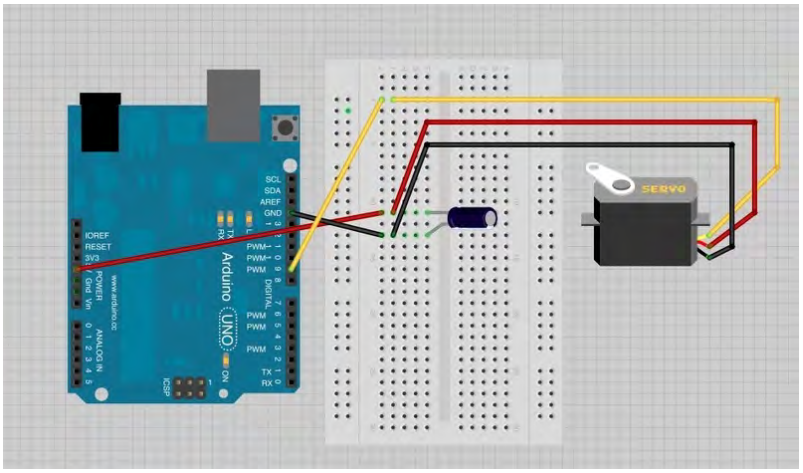

Figura 3.10: Circuito de conexión de un servomotor [25]

3.1.4. Programas

En este apartado se incluyen y explican las partes más significativas de los programas creados. Los programas completos pueden encontrarse comentados en los anexos situados al final del documento.

3.1.4.1. Calibración de los servomotores

Este programa se utilizará para ambos métodos de detección de posición, pero dado que este método fue llevado a cabo en primer lugar, incluiremos el programa en este apartado.

Se trata de un programa simple que nos permitirá, mediante ensayo y error, fijar los ángulos de 0º y 180º de los servomotores, así como su punto medio (que correspondería a un ángulo de 90º); ya que sus valores extremos pueden ser valores superiores a éstos.

Con el objetivo de familiarizarnos con las librerías disponibles en cada entorno para el control de servomotores, comprobaremos los resultados obtenidos con un programa en un entorno diferente de programación.

El programa principal se ha escrito en el entorno *IDE* de *Arduino*. Se trata de un programa muy simple que nos ayudará a conocer los límites de movimiento de nuestros servomotores. Su funcionamiento es el siguiente:

1. El programa se conecta al servomotor e inicia una comunicación serie con el ordenador.

2. En el monitor serie se nos solicita la introducción de un valor para el ángulo deseado. El ángulo se introducirá en milisegundos para obtener una mayor precisión.

3. El servomotor es conducido a la posición deseada mediante la modulación *PWM* del pin al que éste está conectado. Así mismo, el programa informa de la nueva posición del servomotor.

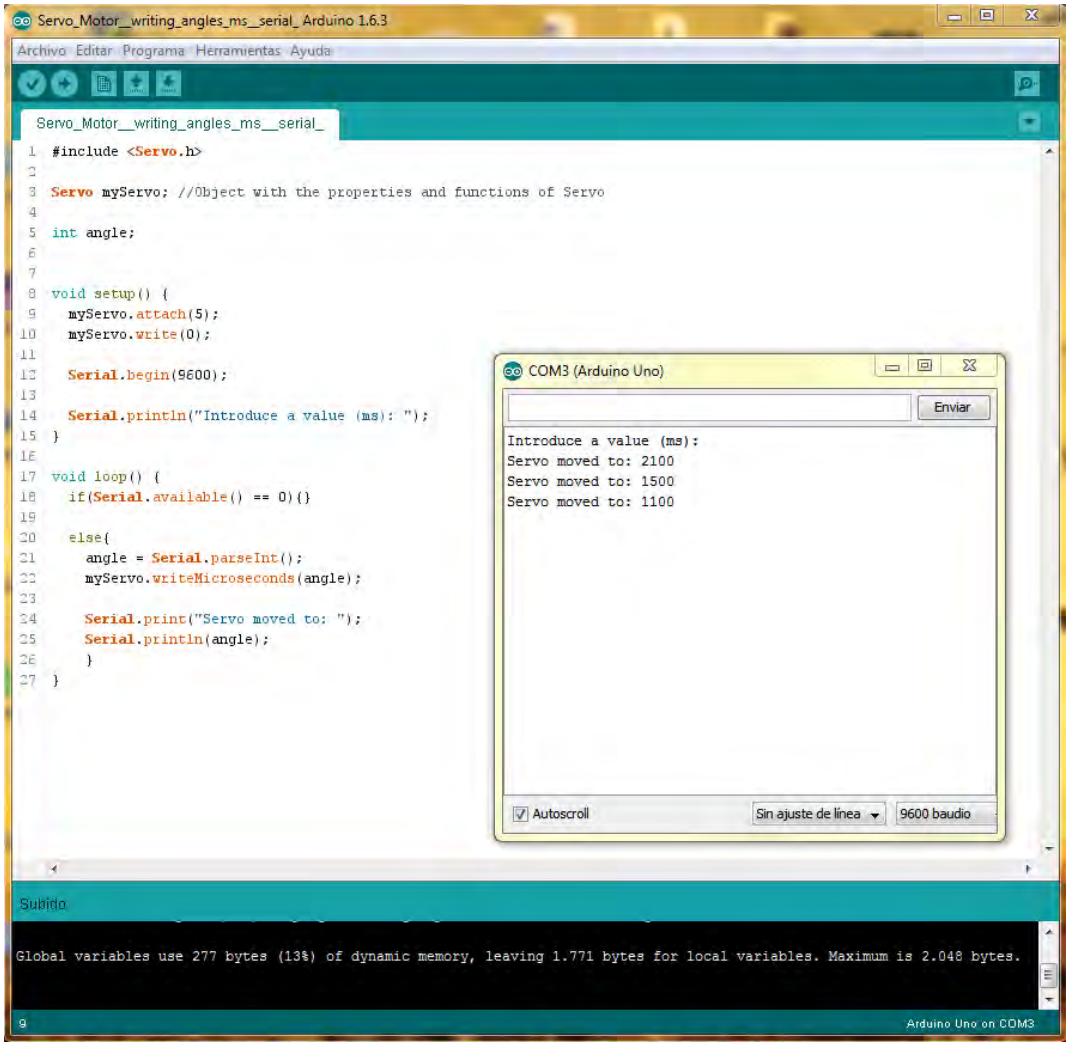

Figura 3.11: Vista general del programa de calibración en *IDE*, en funcionamiento

Este programa será idéntico para ambos servomotores. Su código se puede revisar en el apéndice A.1.1.

Para comprobar los valores obtenidos, utilizaremos código compilado en *MATLAB*. En el interior del programa se establecerán los límites calculados con el programa anterior mediante la siguiente instrucción (valores en milisegundos):

```
servo1 = servo(ard1, 3,
    'MinPulseDuration', 720*10^-6,          % Valor mínimo (0°)
    'MaxPulseDuration', 2300*10^-6);        % Valor máximo (180°)
```

Ahora, introduciendo el valor del ángulo deseado en la consola (comprendido entre 0º y 180º) podremos comprobar si los valores calculados con anterioridad se corresponden con la respuesta angular deseada.

```
while exit == false
        angle_real = input('Introduce desired angle (0-180, x = exit): ',
             's');

        if strcmp(angle_real,'x')
            exit = true;
        else
            angle_real_s = str2double (angle_real);
            angle = angle_real_s/180;

            % Mueve el servo a la posición deseada
            writePosition(servo1, angle);

            % Lee la nueva posición del servo
            current_pos = readPosition(servo1);

            % Muestra la nueva posición en la consola en °
            current_pos = current_pos*180;
            fprintf('Current motor position is %d degrees\n', current_pos
            );
        end

end
```

De nuevo, solicitaremos el ángulo deseado en la consola, moveremos el servo hasta ese ángulo, y mostraremos la nueva posición alcanzada.

3.1.4.2. Medición de posición de la mano y transmisión de la información al servomotor

Para ilustrar el funcionamiento de este método haremos la medición de posición en un eje, concretamente, en el eje 'z' del sensor construido.

El programa de medición se divide en dos partes:

1. La primera parte se encarga de descargar el condensador y preparar los puertos de *Arduino* para llevar a cabo la medición. Además, pone a cero los registros temporales del microprocesador.

```
  for(int i = 2; i < 14; i++) { //Configuramos los puertos de Arduino como
      salidas
    pinMode(i, OUTPUT);
    digitalWrite(i, LOW); //Ponemos los puertos a tierra para descargar el
      condensador
  }

```

```
6  for(int i = 8; i < 11; i++) //Configuramos los pines como entrada
7    pinMode(i, INPUT);
8
9  //Iniciamos la función startTimer(), que pone a cero los registros del
       microprocesador
10 startTimer();
```

2. La segunda parte realiza las medidas de los tiempos de carga. El programa está basado en un método de medición de microsegundos optimizado obtenido de los forums de *Arduino*.

   Utiliza el reloj del microprocesador de la placa para generar un contador temporal, que nos permitirá medir el tiempo de carga del condensador. Al comienzo del programa, definiremos la tasa de refresco:

```
1  #define resolution 8                  //Define la resolución del contador en 8
       bits
2  #define refresh 2 * 1000000 / mains  //Define la tasa de refresco en
       microsegundos: 40000 microsegundos = 40 ms. En binario corresponde al
       número 1001110001000000
```

   En el interior de la variable *time()* del programa incluiremos el código que generará el tiempo de carga del condensador:

```
1  long time(int pin, byte mask) {           //Variable que contiene el tiempo
       en microsegundos que tarda el pin en alcanzar su valor lógico alto
2    unsigned long count = 0, total = 0;
3    while(checkTimer() < refresh) {         //Se cumple mientras el tiempo del
       reloj del microprocesador es inferior a la tasa de refresco
4      pinMode(pin, OUTPUT);
5      PORTB = 0;                            //PORTB sirve para actuar a nivel
       bajo y de forma más rápida sobre el registro de los puertos de Arduino.
       Selecciona los puertos digitales desde el 8 al 13. Esta sentencia los
       fija en su valor lógico bajo para evitar interferencias
6      pinMode(pin, INPUT);
7      while((PINB & mask) == 0)             //PINB lee los valores de los
       puertos comprendidos entre el 8 y el 13, y mask lee el valor del pin que
       se desea leer, es decir, puertos 8, 9 y 10. Por lo tanto, mientras los
       puertos leídos sigan siendo cero, aumentamos el contador
8        count++;
9      total++;
10   }
11   startTimer();                           //Una vez realizada la medida,
       ponemos a cero los registros temporales del microprocesador
12   return (count << resolution) / total;   //Devuelve el valor del tiempo de
       carga medido en microsegundos
13 }
```

   El siguiente fragmento de código detalla las funciones que trabajan con los registros temporales del microprocesador:

```
1  void startTimer() {
2    //Registros temporales del microprocesador
3    timer0_overflow_count = 0; //Contiene el numero de veces que el contador
       Timer #0 se pone a cero
```

```
4     TCNT0 = 0; //El contador Timer #0 es un registro de 8 bits que se
        incrementa una unidad cada 0,004 milisegundos (16MHz del microprocesador
        multiplicado por un preeescalador de 64). Como es de 8 bits, solo
        almacena valores hasta 256, momento tras el cual se pone a cero. Por
        tanto, este registro se pone a cero cada 1,024 ms.
5   }
6
7   unsigned long checkTimer() {
8     return ((timer0_overflow_count << 8) + TCNT0) << 2;      //Devuelve el
        tiempo del reloj del microprocesador en microsegundos. Contiene las veces
         que se ha puesto a cero el registro, mas el valor actual del contador.
        Como se puede observar en la sentencia de la función, el valor del
        contador de puesta a cero del registro Timer #0 se desplaza 10 bits a la
        izquierda, lo que corresponde al número binario 10000000000 = 1024 en
        base decimal. Por lo tanto, cada puesta a cero del contador corresponde a
         1,024 ms. El valor actual del contador representa los microsegundos de
        la medida.
9   }
```

Como se puede observar, el registro temporal se pone a cero cada 1,024 milisegundos, por lo que habría que introducir una corrección al tiempo medido. Sin embargo, para detectar la variación de tiempo de carga del condensador podemos aproximar el tiempo con esta función.

La tasa de refresco definida en el encabezado del programa es de 40ms. Sin embargo, como ya hemos establecido, existirá una desviación con respecto al tiempo real de carga, por lo que si dividimos la tasa de refresco entre la unidad de medida temporal, obtendremos la tasa de refresco real:

$$\frac{40\text{ ms}}{1,024} = 39,0625\text{ ms}$$

En resumen, esto se traduce en que se obtendrá una medida del tiempo de carga del condensador cada 39 ms aproximadamente, lo que supone una frecuencia de muestreo de:

$$f = \frac{1}{T} = \frac{1}{39 \times 10^{-3}} \approx 25\text{Hz}$$

3. La última parte del programa transforma los datos recibidos del sensor en valores válidos para los servomotores, y envía la orden de movimiento.

```
1   angle = map(time(9, B00000010), 9670, 10300, 720, 2300); //Transformamos los
        valores recibidos a un rango válido para el servomotor
2
3   servo1.writeMicroseconds(angle); //Movemos el servomotor al punto indicado
        por el sensor
```

La función *map()* requiere una atención especial en este caso, ya que es la encargada de transformar los datos recibidos del sensor en valores que el servomotor puede manejar. La sintaxis de la función es la siguiente:

```
1   map(value, fromLow, fromHigh, toLow, toHigh)
```

Como se puede observar, esta función toma como valores de entrada los límites del rango original y los del nuevo. De esta forma, escala los valores al rango que se necesite.

En este caso, se introduce una aproximación de los valores máximo y mínimo medidos para el tiempo de carga, y se transforman al rango de movimientos posibles para el servomotor.

3.1.5. Resultados

El comportamiento del sistema se puede observar en el siguiente vídeo:

Con el objetivo de definir el comportamiento del sistema se ha llevado a cabo una recopilación de medidas tomadas a través del monitor serie de *Arduino*, con las que se han constituido unas curvas de calibración.

Para poder tomar estas medidas se ha introducido una espera de 2 segundos en el interior del programa de medición para facilitar la extracción de los datos, gracias a la función *delay()*.

3.1.5.1. Mano paralela al sensor

En este primer caso, se han tomado las medidas posicionando la mano paralela a la placa del sensor. Se han obtenido los siguientes resultados:

Cuadro 3.1: Medidas tomadas con la mano paralela a la placa

Distancia (cm)	**Tiempo (μs)**	**Capacidad ($\pm 0,0007$ nF)**
∞	9452	7,0013
10,0	9507	7,0422
9,0	9524	7,0548
8,0	9544	7,0696
7,0	9550	7,0741
6,0	9558	7,0800
5,0	9573	7,0909
4,0	9583	7,0987
3,0	9734	7,2102
2,0	9824	7,2772
1,5	9914	7,3433
1,0	9928	7,3537
0,5	10088	7,4726
0,2	10179	7,5398

Los datos de capacidad han sido calculados haciendo uso de la ecuación (9).

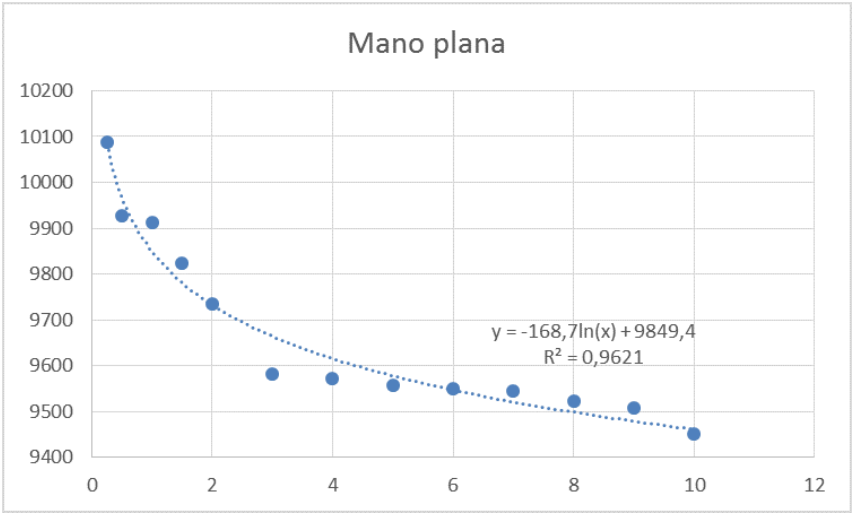

Figura 3.12: Curva de calibración del sensor con la mano paralela a la placa

Como se puede observar, la relación entre la distancia de la mano a la placa y el tiempo de carga está definida por una ecuación logarítmica. Esto demuestra que la capacidad aumenta cuando la mano está cerca del sensor, pero a distancias mayores las variaciones son muy pequeñas.

En un condensador ideal de placa paralelas posee dos planos infitos paralelos, separados una distancia muy pequeña. En este caso, la capacidad se define como:

$$C = \varepsilon_0 \varepsilon_r \times \frac{A}{d} \tag{10}$$

siendo:

- ε_0: permitividad del vacío
- ε_r: permitividad relativa del dieléctrico existente entre las placas
- A: área de las placas del condensador
- d: distancia existente entre las placas del condensador

Si el dieléctrico es el aire, como en el caso estudiado, $\varepsilon_r \approx 1$. Se puede decir entonces que la capacidad mantiene una relación lineal con la distancia entre las placas. Sin embargo, en el caso estudiado las medidas demuestran que en el sensor construido esta relación es logarítmica. Ésto ocurre porque nuestra mano no tiene la misma morfología que la placa del sensor.

En resumen, si se sitúa la mano muy cerca del sensor se está creando un condensador cercano al ideal. Por el contrario, si se coloca la mano a mayor distancia de la placa, la respuesta del condensador difiere de la esperada.

3.1.5.2. Otras posiciones

A continuación se añaden los resultados de las mediciones tomadas con la mano colocada a 45º y a 90º con respecto a la placa del sensor.

Cuadro 3.2: Medidas tomadas con la mano a 45 y a 90º

Mano a 45º			**Mano a 90º**	
Distancia (cm)	**Tiempo (μs)**	**Capacidad ($\pm 0,0007$ nF)**	**Tiempo (μs)**	**Capacidad ($\pm 0,0007$ nF)**
∞	9449	6,9989	9448	6,9985
10,0	9513	7,0469	9500	7,0370
9,0	9513	7,0467	9515	7,0480
8,0	9516	7,0489	9526	7,0559
7,0	9524	7,0544	9540	7,0667
6,0	9536	7,0633	9545	7,0700
5,0	9545	7,0702	9546	7,0713
4,0	9555	7,0776	9555	7,0774
3,0	9582	7,0978	9574	7,0917
2,0	9633	7,1356	9639	7,1396
1,5	9664	7,1585	9647	7,1457
1,0	9755	7,2259	9721	7,2007
0,5	9812	7,2678	9784	7,2476
0,2	9959	7,3767	9914	7,3437

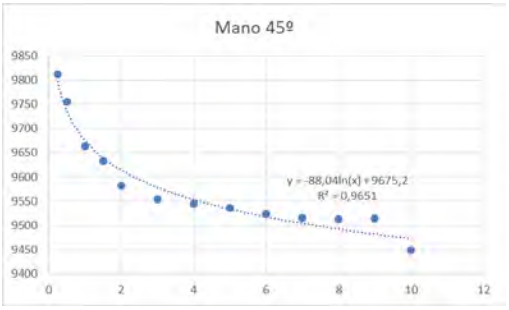

Figura 3.13: Curva de calibración de 45º

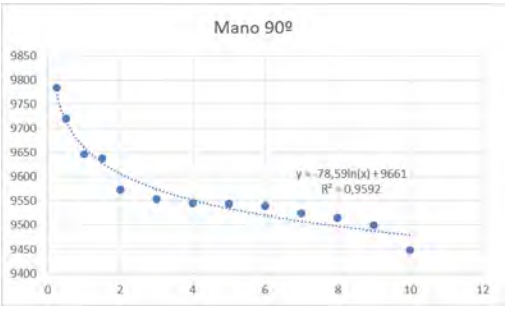

Figura 3.14: Curva de calibración de 90º

Se puede observar que en estos casos también aparece una relación logarítmica entre el tiempo y la distancia mencionada con anterioridad.

Para comprobar el efecto que tiene la variación de la forma de la mano en la respuesta del sensor se han comparado las tres gráficas anteriores:

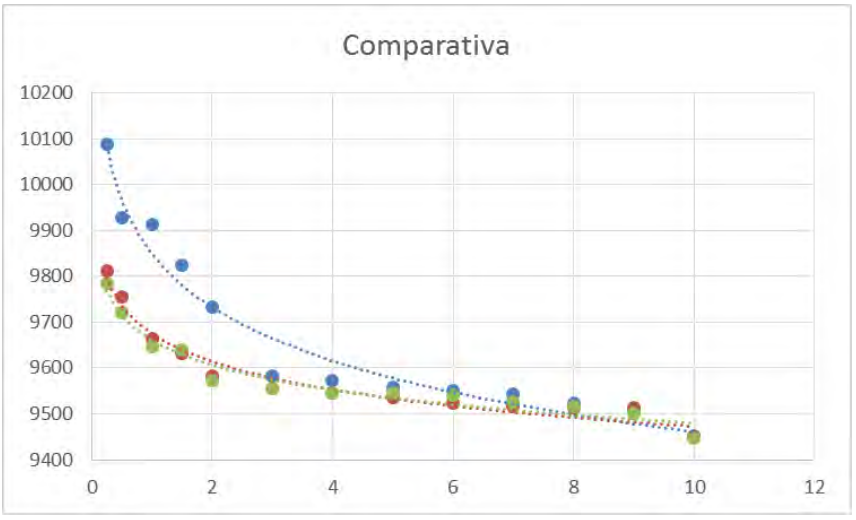

Figura 3.15: Comparativa de la respuesta del sensor en función de la posición de la mano

De la figura 3.15 se pueden extraer varias conclusiones:

- Si la mano está situada a distancias mayores de 6 cm, el sensor se comporta de forma similar en los tres casos estudiados, por lo que se puede establecer que la mayor parte del rango de detección se encuentra a una distancia de entre 0 y 6 cm de la placa del sensor. Este rango se denominará 'umbral de detección'.

- Dentro del umbral de detección, se puede observar que colocando la mano paralela a la placa se consiguen mayores variaciones del tiempo (se trata de una consecuencia directa de lo establecido con anterioridad. A distancias cortas, si se coloca la mano paralela a la placa del sensor, se estará generando un condensador de comportamiento cercano al ideal). Como consecuencia, el sensor poseerá mayor precisión a la hora de estimar la posición de la mano, ya que tiene más información con la que trabajar.

Por el contrario, el sensor es incapaz de detectar la diferencia entre la mano colocada a 45º y la colocada a 90º. Como consecuencia, las curvas de calibración son prácticamente idénticas.

3.1.6. Conclusiones y adecuación del método

Como se puede observar en el vídeo mostrado con anterioridad, la capacidad de detección del sistema es limitada. El rango de detección de proximidad se restringe a una separación máxima de la mano de 10 cm.

La precisión obtenida no es muy alta, ya que el servomotor realiza movimientos un tanto bruscos, y varía levemente su posición aunque nuestra mano esté inmóvil. Esto es debido a que a grandes distancias del sensor, la variación del tiempo medido es muy pequeña, como describe la ecuación logarítmica de las curvas de calibración. Estas fluctuaciones en las medidas podrían dificultar el control preciso de la antena.

Así mismo, el sistema es poco intuitivo, ya que no existe retroalimentación ante el movimiento de la mano, y esta se encuentra en una posición un tanto forzada al tener que ser mantenida en alto, lo que resultaría muy incómodo a la hora de ser utilizado durante largos periodos de tiempo.

Sin embargo, este método destaca por su simplicidad de construcción y su bajo coste, ya que no necesita más recursos que el cubo de detección y los cables apantallados.

3.2. Método 2º. Acelerómetro

Como se ha explicado con anterioridad en la sección 1.3, los acelerómetros son muy usados hoy en día en distintos ámbitos, especialmente en la detección de posición y orientación de un objeto.

De forma más especializada, se han utilizado para el reconocimiento de gestos, aunque hoy en día, debido a factores de coste y simplicidad, cada vez se puede observar de forma más abundante estos sistemas aplicados a situaciones de la vida cotidiana, como teléfonos inteligentes, tabletas, etc.

En este trabajo, se utilizará un acelerómetro triaxial para medir la gravedad estática, y controlar con la información extraída de dos de sus ejes de medida los servomotores ya mencionados.

El objetivo de este método es conseguir un control intuitivo del mecanismo deseado, de manera que resulte natural, tenga una respuesta correcta y una sensibilidad adecuada para la aplicación estudiada.

3.2.1. Principio de funcionamiento

La primera impresión sobre este método anuncia que su funcionamiento será más complicado que el anterior. Sin embargo lo cierto es que gracias a las técnicas de producción *MEMS*, se dispone de un acelerómetro que va a realizar prácticamente todos los trabajos necesarios.

Por lo expuesto, el funcionamiento de este método queda resumido en los siguientes pasos:

1. El acelerómetro mide la inclinación existente, mediante la variación de capacidad producida en su interior como consecuencia de la aceleración sufrida, como se ha explicado en la sección 2.4.
2. La información de medida se extrae mediante el establecimiento de una conexión serie creada a través de un bus SPI.
3. Los datos extraídos del acelerómetro se transforman en valores pertenecientes al rango de trabajo de los servomotores.
4. Un programa implementado en *MATLAB* lee los valores transmitidos a través de la conexión serie y muestra en una interfaz gráfica los cambios de orientación realizados.

3.2.2. Conexiones

El circuito necesario para implementar este método sa basa únicamente en conectar el acelerómetro (ADXL345) a la placa *Arduino*. El circuito de conexión de los servomotores será idéntico al del método 1, mostrado en la sección 3.1.3.2.

Como se indica en la sección 2.4.3.3, se utilizará el protocolo SPI para establecer la comunicación con el ADXL345. Se fijará la configuración del reloj definida por ‘SPI Mode 3’, con el reloj en reposo en su valor lógico alto y la lectura del bus en el flanco de bajada.

Arduino Pin	ADXL345 Pin
10	CS
11	SDA (MISO)
12	SDO (MOSI)
13	SCL (SCK)
3V3	VCC
Gnd	GND

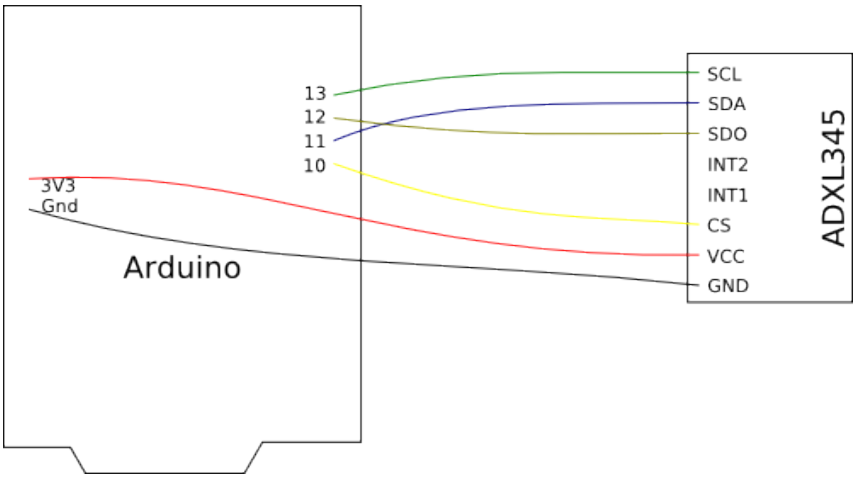

Figura 3.16: Conexión del ADXL345 a *Arduino* [24]

3.2.3. Programas

3.2.3.1. Código en Arduino

El programa de control del ADXL345 se basa en un bus de comunicación SPI establecido entre el acelerómetro y *Arduino*. Se utilizarán los registros del ADXL345 para configurarlo en función de nuestras necesidades. En este caso, debido a la extensión del programa, se dividirá el código en tres partes diferenciadas: la función *setup()*, la función *loop()* y las funciones creadas de forma específica.

1. El el interior de la función setup(), se definirá la configuración deseada para el acelerómetro y se designarán los pines de conexión del acelerómetro y de los servos.

```
1  void setup(){
2    SPI.begin();  //Inicia la comunicación SPI
3    SPI.setDataMode(SPI_MODE3); //Configura el modo SPI para el ADXL345
4    Serial.begin(9600); //Abre una comunicación a través del puerto serie para
       mostrar los datos recibidos en el monitor serie
5
6    pinMode(CS, OUTPUT);  //Configura el SS (chip select) de Arduino como
       salida, que ha sido definido con anterioridad
7    digitalWrite(CS, HIGH); //Antes de comenzar la comunicación, se debe fijar
       el pin de Chip Select en su valor lógico alto
8
9    writeRegister(DATA_FORMAT, 0x00); //Configura el ADXL345 para un rango de
       sensibilidad de +/- 2G
10
11   writeRegister(INT_ENABLE, 0xE0);  //Activa el control de interrupción
12
13   writeRegister(POWER_CTL, 0x08); //Pone el ADXL345 en 'Measurement Mode'
       para que comience a tomar medidas
14
```

```
15      //Establece los puertos de control para los servos y los coloca en su punto
            medio
16      servoX.attach(5);
17      servoX.writeMicroseconds(1500);
18      servoY.attach(6);
19      servoY.writeMicroseconds(1400);
20    }
```

Como se puede observar, la configuración del acelerómetro se lleva a cabo actuando directamente sobre los registros del mismo a través de la función *writeRegister()*, cuyo funcionamiento se detallará más adelante.

2. En el interior de la función *loop()*, se extraen los datos de medida de inclinación del acelerómetro y los se convierten a valores de trabajo de los servomotores.

```
1
2       //Convierte los valores medidos a valores de trabajo de los servomotores
3       angleX = map(x, -255, 255, 720, 2300);
4       servoX.writeMicroseconds(angleX);
5       angleY = map(y, -255, 255, 550, 2280);
6       servoY.writeMicroseconds(angleY);
7
8       // Mediante la lectura de 6 bytes de datos empezando por el registro DATAX0
            obtiene los valores de la aceleración en los ejes 'x', 'y' y 'z' medidos
            por el ADXL345
9       //Los resultados de la operación de lectura se almacenan en un buffer
            creado por nosotros: 'values[]'
10      readRegister(DATAX0, 6, values);
11
12      //El ADXL345 proporciona valores de aceleración con una resolución de 10
            bits (para una configuración de +/- 2g), pero los almacena en 2 bytes (8
            bits cada byte). Por lo tanto, se deben combinar dos bytes para obtener
            la medida completa de cada eje
13      //El valor del eje 'x' se almacena en las componentes 0 y 1 del buffer (
            values[0] y values[1])
14      x = ((int)values[1]<<8)|(int)values[0];
15      //El valor del eje 'y' se almacena en las componentes 2 y 3 del buffer (
            values[2] y values[3])
16      y = ((int)values[3]<<8)|(int)values[2];
17      //El valor del eje 'x' se almacena en las componentes 4 y 5 del buffer (
            values[4] y values[5])
18      z = ((int)values[5]<<8)|(int)values[4];
19
20      //Muestra los valores medios a través de la comunicación serie
21      Serial.print(x);
22      Serial.print(' ');
23      Serial.println(y);
24
25      //El programa espera 50ms para tomar la siguiente medida. Esto permite
            obtener una tasa de refresco de aproximadamente 20Hz.
26      delay(50);
```

Se puede observar que al comienzo del fragmento de código anterior se utiliza de nuevo la función *map()* de *Arduino* para transformar el rango de valores del acelerómetro en aquel que nos permita controlar los servomotores.

Como ya se indicó en la sección 2.4.4, el ADXL345 ajusta la resolución de las medidas que proporciona en función del rango de medida escogido: 13 bits para medidas de +/- 16g, 12 bits para medidas de +/- 8g, 11 bits para medidas de +/- 4g, y por último, 10 bits para medidas de +/- 2g.

Una resolución de 10 bits supone que la medida se expresará en valores comprendidos entre 0 y 1024 en el sistema decimal ($2^{10} = 1024$). Dado que el número binario está codificado en complemento a dos (incluye números negativos), las medidas comprenderán números entre -512 y +512. En nuestro caso, en la inclinación máxima alcanzable en un eje (90^{o}) se estará sometiendo el acelerómetro a una aceleración de 1g en dicho eje, como muestra la figura 2.24.

Partiendo del hecho de que el acelerómetro está configurado en un rango de +/- 2g, se puede deducir que el rango de medida para detectar las inclinaciones estará comprendido entre -255 y +255, es decir, la mitad de la resolución total del acelerómetro en este modo. Estos son los valores que se pueden encontrar en el argumento de la función *map()* de este método.

Al restringir las medidas a la mitad de la resolución del acelerómetro se estarán utilizando únicamente las aceleraciones inferiores o iguales a 1g.

3. Las funciones *writeRegister()* y *readRegister()* se encargan de acceder al registro del ADXL345 y realizar la configuración y lectura de valores del mismo.

writeRegister():

```
1  //Esta función escribirá las ordenes correspondientes en los registros del
       ADXL345.
2  //Parámetros:
3  //   char registerAddress - Contiene el registro de destino
4  //   char value - Contiene el valor que se debe escribir en el registro
       especificado
5  void writeRegister(char registerAddress, char value){
6
7    digitalWrite(CS, LOW);  //Fijam el pin SS (chip select) en su valor lógico
       bajo para comenzar la transmisión SPI
8    SPI.transfer(registerAddress);  //Transfiere la dirección del registro de
       destino
9    SPI.transfer(value);  //Transfiere el valor que se quiere escribir en el
       registro
10   digitalWrite(CS, HIGH); //Fija el pin SS en su valor lógico alto para
       finalizar la transmisión SPI
11 }
```

readRegister():

```
1  //Esta función leerá un número concreto de registros, empezando por la
       dirección especificada, y almacenará sus valores en un buffer
2  //Parámetros:
3  //   char registerAddress - Contiene la dirección del registro a partir de la
       cuál se comienza la lectura
4  //   int numBytes - Contiene en número de registros que se deben leer
5  //   char * values - Contiene un puntero que indica el buffer en el que se
       deben almacenar los datos
6  void readRegister(char registerAddress, int numBytes, unsigned char * values)
       {
```

```
7       //Como se especifica en el Datasheet del ADCL345, para realizar una
          operación de lectura, el bit más significativo de la dirección del
          registro debe ser fijado
8       char address = 0x80 | registerAddress;
9       //Si se realiza una operación de multi-lectura, el bit número 6 también
          debe ser fijado
10      if(numBytes > 1)address = address | 0x40;
11
12      //Fija el pin SS en su valor lógico bajo para comenzar la transmisión SPI
13      digitalWrite(CS, LOW);
14      //Transfiere la dirección del primer registro que se debe leer
15      SPI.transfer(address);
16      //Continúa leyendo registros hasta que se alcanza el número establecido, y
          los almacena en el buffer designado para ello (values[])
17      for(int i=0; i<numBytes; i++){
18        values[i] = SPI.transfer(0x00);
19      }
20      //Fija el pin SS en su valor lógico alto para finalizar la transmisión SPI
21      digitalWrite(CS, HIGH);
22    }
```

3.2.3.2. Interfaz de MATLAB

El aspecto de la interfaz elaborada en *MATLAB* se puede observar en la figura 3.17.

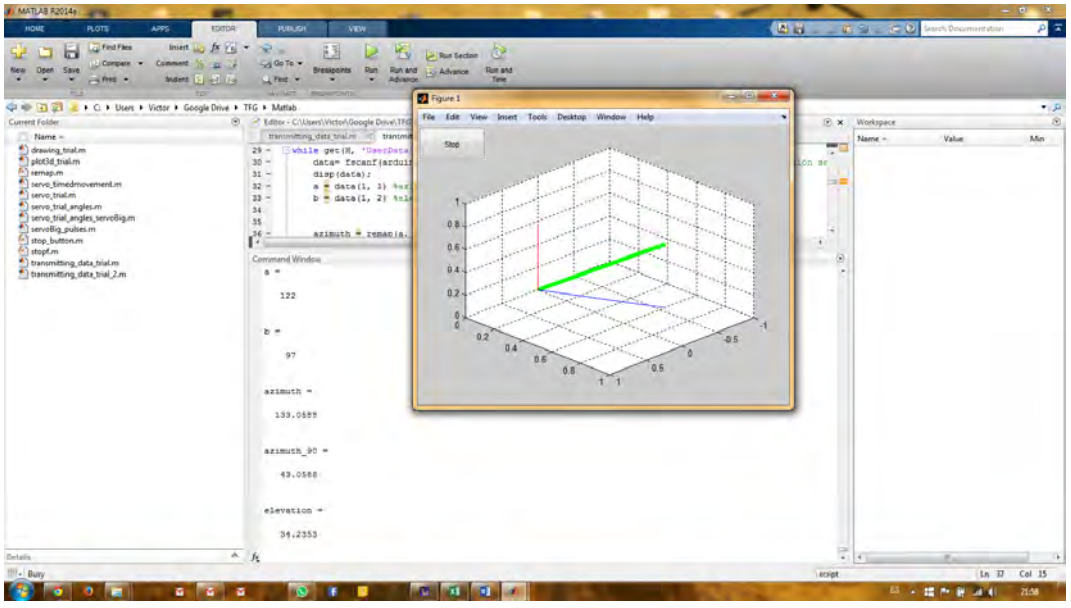

Figura 3.17: Ejes coordinados

En la interfaz, se distinguen tres vectores de colores:

- El vector verde representa la orientación real de el objeto a controlar.
- El vector azul representa la proyección sobre la horizontal de la línea verde. Corresponde al ángulo de azimut medido por el acelerómetro.
- El vector rojo representa la elevación medida por el acelerómetro.

Los dos últimos vectores buscan facilitar el reconocimiento de la orientación del objeto. También se puede observar que en la ventana de comandos se muestran los valores de salida del acelerómetro así como los ángulos de acimut y elevación calculados. Gracias a la propiedad de *MATLAB* que nos permite mostrar resultados intermedios, se han podido realizar los experimentos descritos en la sección 3.2.4. A grandes rasgos, el programa escrito en *MATLAB* toma las mediciones del acelerómetro y las traduce a una orientación en un espacio de tres dimensiones. El funcionamiento se describe a continuación:

1. El programa toma las medidas del acelerómetro, y las transforma en valores angulares. Este proceso se realiza gracias a una función que se ha denominado *remap()*. Esta función se encarga de transformar un rango de valores de entrada a otro rango de salida, conociendo los valores máximo y mínimo de cada rango. Su funcionamiento es idéntico al de la función *map()* de *Arduino*:

```
function x = remap(value, fromLow, fromHigh, toLow, toHigh)

        %Expresión matemática que transforma los rangos
    x = (value - fromLow) * (toHigh - toLow) / (fromHigh - fromLow) +
     toLow;

end
```

2. Una vez se dispone de los valores angulares, se conocen las coordenadas esféricas del objeto que se está orientando. A través de una transformación matemática, se pueden obtener las coordenadas cartesianas de un objeto partiendo de sus coordenadas esféricas. Dicho proceso está descrito por las ecuaciones:

$$x = R \times cos(\text{elevacion}) \times cos(\text{acimut})$$

$$y = R \times cos(\text{elevacion}) \times sin(\text{acimut})$$

$$z = R \times sin(\text{elevacion})$$

3. Con un origen de referencia definido y las coordenadas cartesianas del objeto es posible establecer un vector que posea la dirección que se conseguiría con esa inclinación específica del acelerómetro. Este vector es el que se puede observar en color verde en la interfaz.

Mediante esta serie de procesos se consigue transferir las medidas del acelerómetro a un espacio en tres dimensiones, en el que es posible observar de forma virtual la orientación del objeto. Este hecho permite obtener una idea más comprensible de cuál sería el funcionamiento del sistema una vez construido.

El código completo del programa se puede revisar en el apéndice A.2.2.

3.2.4. Resultados

En los siguientes vídeos se puede observar el comportamiento del sistema y el funcionamiento de la interfaz de *MATLAB*.

Las medidas obtenidas del acelerómetro son valores que a simple vista no tienen sentido numérico. Sin embargo, se puede definir una escala de transformación de los valores medidos a unidades de gravedad (g), donde $1g = 9{,}81\ \mathrm{m} \times\ \mathrm{s}^{-2}$. La escala mencionada se define como:

$$Escala = \frac{Rango\ de\ medida}{Resolucion\ de\ medida}$$

En este caso, se ha configurado el acelerómetro en un rango de medida de +/- 2g, constituyendo un rango de 4g; y la resolución de medida del acelerómetro es de 10 bits. La escala buscada se obtendrá dividiendo el rango total entre el conjunto de números que se pueden representar en 10 bits:

$$Escala = \frac{4}{2^{10}} = 0{,}0039$$

Por lo tanto, multiplicando los valores que aparecen en el monitor serie por el factor de escalado indicado a continuación se obtendrán las medidas en unidades equivalentes a la aceleración de la gravedad.

Por ejemplo, para una inclinación de 90º:

- Valor en el monitor serie: $a_m = 255$
- Valor de la medida en g: $a_g = a_m \times Escala = 0{,}9945 \approx 1g$

3.2.4.1. Giros lentos, suaves y progresivos

Este tipo de giros será el más habitual en el control de orientación del dispositivo. se procederá a la inclinación del acelerómetro de forma suave y progresiva hasta alcanzar la posición deseada. En este caso, los servomotores alcanzan sin problemas la posición solicitada. Dado que se trata de un giro lento, los servomotores nunca alcanzarán su máxima velocidad; girarán de forma lenta conforme se varía la posición solicitada por el programa. En el cuadro 3.3 se incluyen las medidas realizadas con este tipo de giros, así como el cálculo del ángulo y la intensidad de la gravedad a la que está sometido el acelerómetro en ese momento.

Cuadro 3.3: Valores medios de salida medidos al someter el acelerómetro a giros suaves

Inclinación real (º)	**Inclinación medida (eje x)** ($\pm 0,4$**º**)	**Valor de salida (eje x)**	**Intensidad de la gravedad** ($\pm 0,04$ **g**)	**Inclinación medida (eje y)** ($\pm 0,4$**º**)	**Valor de salida (eje y)**	**Intensidad de la gravedad** ($\pm 0,04$ **g**)
0,0	7,4	21	0,08	4,4	13	0,05
30,0	30,2	86	0,33	30,7	87	0,34
45,0	45,3	128	0,50	45,5	129	0,50
60,0	59,8	170	0,66	60,5	172	0,67
90,0	100,6	286	1,12	100,1	285	1,11
-30,0	-29,8	-85	-0,33	-30,3	-85	-0,33
-45,0	-45,4	-129	-0,50	-45,4	-127	-0,49
-60,0	-60,0	-170	-0,66	-60,5	-169	-0,66
-90,0	-90,0	-253	-0,98	-91,6	-260	-1,01

Los valores de la inclinación se han calculado a través de *MATLAB*, con la función implementada para dicho fin. La intensidad de la gravedad se ha calculado multiplicando el valor medido por el acelerómetro por la escala especificada en la ecuación de la página 62. Los valores de salida de ambos ejes están referidos a los valores sin tratar transmitidos por el acelerómetro. Con el objetivo de comprobar la repetitividad, las medidas se han repetidos dos veces y se ha calculado la media. El resto de medidas pueden encontrarse en el apéndice C.

La figura 3.18 contiene una gráfica que compara los valores reales de inclinación del acelerómetro con las medidas indirectas realizadas a través de *MATLAB*.

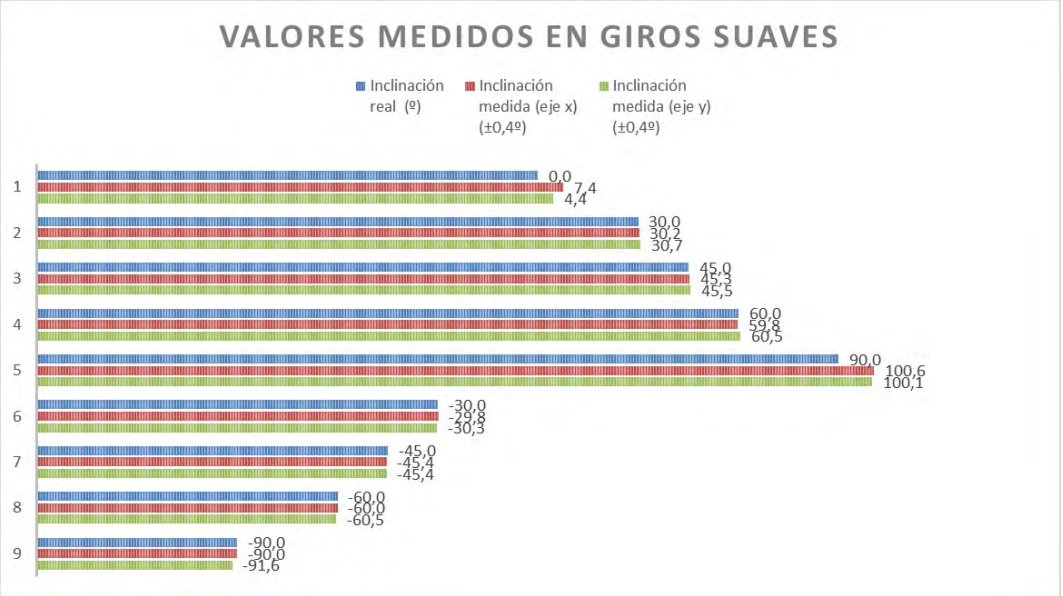

Figura 3.18: Comparativa entre el valor real de inclinación y el valor medido

Cabe destacar que el acelerómetro es altamente preciso en las medidas de inclinación. Sin embargo, en ambas figuras se pueden observar dos fenómenos que no concuerdan con la exactitud del acelerómetro en la mayor parte de su rango de medida:

1. En las medidas intermedias, el acelerómetro es muy preciso. Se observa que para 30º, 45º, y 60º la medidas son prácticamente idénticas a la realidad.

   Sin embargo, en los extremos, ocurre algo muy distinto. Cuando uno de los ejes evaluados (‘x’ e ‘y’) alcanza una inclinación de 90º con el plano horizontal, el acelerómetro pierde precisión, llegando a medir una gravedad estática superior a 1g. Las variaciones son pequeñas, del orden de 0,1g, pero se traducen en una desviación de medición de casi 10º. Por el contrario, al alcanzar los -90º, ésta desviación es mucho menor.

   Se puede concluir que el acelerómetro trabaja con mayor precisión para inclinaciones negativas de sus ejes.

2. En reposo, se observa que la salida que ofrece el acelerómetro no corresponde a una inclinación de 0º. Ésto puede estar causado por dos motivos:

   - La mesa de trabajo no es una referencia de inclinación de 0º.
   - Los pines del acelerómetro no están rectos. Al conectarlo a una ’Protoboard’, como es el caso de este trabajo, la inclinación de reposo se podría ver afectada por este tipo de defectos.

3.2.4.2. Giros rápidos y bruscos

Cuando se somete el acelerómetro a giros bruscos, los servomotores tratan de seguir el movimiento. Sin embargo, si el cambio de inclinación del acelerómetro supera la velocidad máxima del servo, éste girará intentando alcanzar el valor que el programa le pide sin éxito.

Si se evalúa lo que ocurre ante un cambio brusco de inclinación:

1. El servomotor intenta alcanzar la posición pedida.
2. Si antes de que lo consiga, se invierte la inclinación del acelerómetro, el servomotor interrumpe su giro en la dirección inicial e intenta alcanzar la nueva posición.
3. Si tras un giro brusco, se estabiliza el acelerómetro, el servomotor alcanzará la posición solicitada por el programa girando a su máxima velocidad y pasará al reposo.

Con conocimiento de las velocidades máximas de los servomotores utilizados, es posible calcular el cambio de inclinación máxima que pueden seguir. La velocidad angular del giro se calcula como la variación del ángulo entre el tiempo en el que se realiza:

$$\omega = \frac{d\Theta}{dt} = \frac{\Theta_f - \Theta_o}{t_f - t_o}$$

siendo:

- Θ_f: ángulo final
- Θ_o: ángulo inicial
- t_f: tiempo final
- t_o: tiempo inicial

Cuadro 3.4: Velocidades angulares máximas de los servomotores

	Velocidad Hoja de Características v (s/60º)	**Velocidad angular máxima ω (rad/s)**
S2309S	0,12	500
SANWA	0,2	300

Las velocidades máximas del cuadro 3.4 se consiguen alimentando a los motores con $4{,}8$ V ≈ 5 V (alimentación procedente del pin de *Arduino*).

Por lo tanto, para cambios de inclinación que se produzcan a más de 500 rad/s, el servomotor S2309S no sería capaz de alcanzar la posición solicitada. El servomotor de SANWA no sería capaz de seguir cambios que se produzcan a más de 300 rad/s.

Expresado de otra forma, para un cambio brusco de 0º a 60º, el S2309S tardaría 0,12 s en alcanzar la nueva posición, mientras que el SANWA tardaría 0,2 s.

3.2.4.3. Giros con el eje 'z' en horizontal

En este caso se estudia qué ocurre si se coloca el eje 'z' de forma horizontal, se somete uno de los ejes del acelerómetro estudiados ('x' o 'y') a una inclinación total, es decir, 90º, y se intenta crear una variación en el otro eje.

Imaginemos que se gira el acelerómetro de forma que el eje 'z' quede horizontal y el eje 'x' apunte hacia abajo.

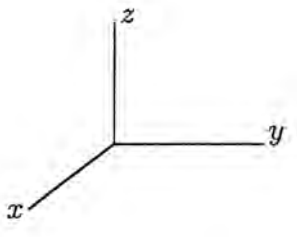

Figura 3.19: Ejes coordinados

Si en esta posición se rota de forma suave el acelerómetro sobre el eje x, no se está generando un cambio notable en la aceleración de ninguno de los ejes, por lo que los servomotores apenas varían su posición.

Si por el contrario, se gira sobre el eje 'z', se estará disminuyendo la aceleración sobre el eje 'x' (que antes apuntaba hacia abajo) y aumentándola sobre el eje 'y'. El acelerómetro se comporta de la forma esperada y el programa disminuye el ángulo del servomotor correspondiente al eje 'x' y aumenta el del correspondiente al eje 'y'.

El cuadro 3.5 denota los valores medidos en *MATLAB* durante la ejecución de este experimento:

Cuadro 3.5: Valores medios medidos con el eje 'z' en posición horizontal

Inclinación referencia (eje x) (º)	**Inclinación medida (eje x) ($\pm 0,4$º)**	**Valor de salida (eje x)**	**Inclinación referencia (eje y) (º)**	**Inclinación medida (eje y) ($\pm 0,4$º)**	**Valor de salida (eje y)**
90,0	100,9	286	0,0	0,6	2
60,0	80,2	223	30,0	67,8	190
45,0	60,7	171	45,0	87,5	244
30,0	30,7	94	60,0	98,5	177
0,0	0,3	2	90,0	100,6	288
-90,0	-88,9	-253	0,0	-1,5	-2
-60,0	-82,7	-232	-30,0	-31,0	-85
-45,0	-73,7	-209	-45,0	-46,4	-133
-30,0	-50,3	-146	-60,0	-70,6	-203
0,0	-0,4	-2	-90,0	-91,3	-256

Como se puede observar en la tabla, la precisión del acelerómetro es bastante pobre en esta situación, dando lugar a grandes desviaciones en las medidas. Las figuras 3.20 y 3.21 muestran este fenómeno de forma gráfica.

Como se ha indicado con anterioridad, en esta situación el eje ‘z’ estaría colocado de forma horizontal, y se pivotaría sobre él aumentando y disminuyendo respectivamente los ángulos en los otros dos ejes.

Debido a las desviaciones observadas, se puede concluir que la precisión del acelerómetro es mucho mayor cuando el eje ‘z’ forma un ángulo con la horizontal mayor de 0º; como por ejemplo, en la sección 3.2.4.1, donde se ha observado que las medidas son mucho más exactas.

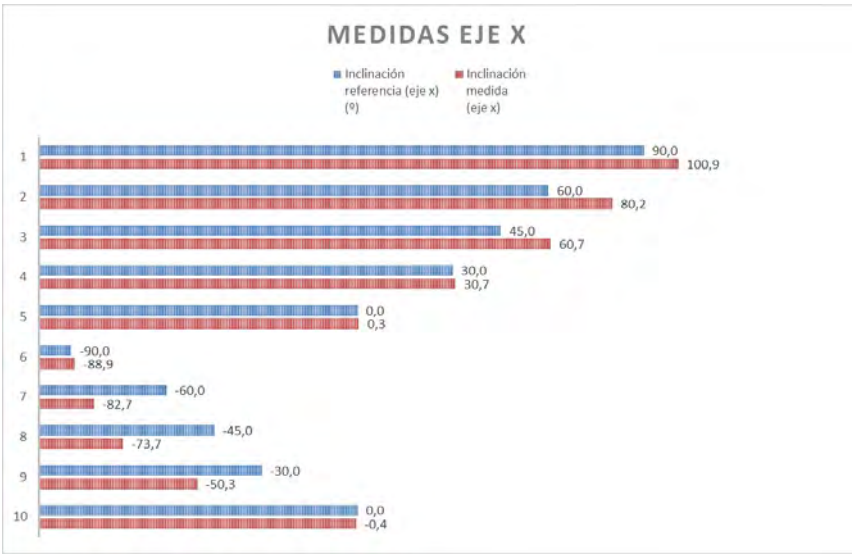

Figura 3.20: Desviaciones de medida en el eje ‘x’

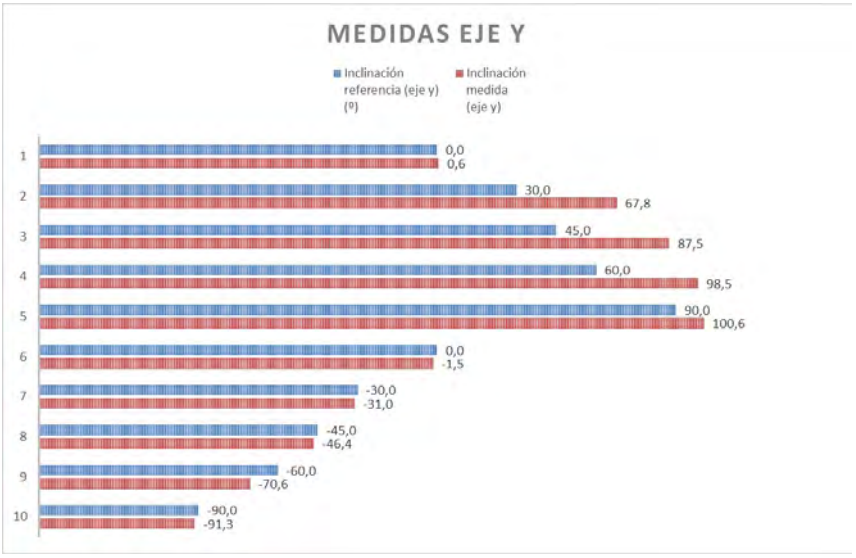

Figura 3.21: Desviaciones de medida en el eje ‘y’

Por lo aquí expuesto, para la medida de aceleraciones en los ejes ‘x’ e ‘y’ se recomienda la utilización del acelerómetro colocado de forma horizontal, con la cara que aparece en la figura 2.26 posicionada hacia arriba.

3.2.5. Conclusiones y adecuación del método

Como se puede observar en los vídeos mostrados anteriormente, el sistema responde de forma precisa ante nuestros gestos.

Se puede observar que, cuando el acelerómetro se encuentra en reposo, los servomotores no cambian su posición, lo que indica que la estabilidad del sistema es alta, y no existen oscilaciones en las medidas. Los servomotores responden de forma suave a nuestros movimientos, alcanzando los posiciones solicitadas con rapidez y precisión. Esto asegura que en una situación en la que se desee mantener la antena estática, no existan movimientos involuntarios de la misma.

Mediante diversos experimentos se ha determinado la respuesta del sistema ante diversas situaciones, y se ha formulado una recomendación de cara a la utilización del acelerómetro. El objetivo de éste trabajo es conseguir una orientación precisa mediante giros suaves y progresivos. Se ha demostrado que el sistema cumple esta función con alta precisión (estudio llevado a cabo en la sección 3.2.4.1).

Dado que el acelerómetro tiene un tamaño muy reducido, sería muy sencillo adaptarlo a un guante o un artefacto similar. Esto compondría un sistema poco invasivo que se integraría con el usuario, y permitiría un control sencillo e intuitivo de los servomotores.

Los posibles avances en el desarrollo de este método se detallarán en la sección de Desarrollos futuros.

Capítulo 4

Conclusiones

A continuación se presenta la discusión final, así como los posibles desarrollos futuros de este trabajo.

4.1. Discusión final

El presente documento describe la construcción de un proceso de orientación remota de sistemas actuado mediante gestos. Con ánimo de establecer una base para poder valorar el rendimiento del proceso creado, se han estudiado dos métodos posibles de realización de la regulación.

El primer método fue desarrollado partiendo de un sensor capacitivo. Se ha llevado a cabo una descripción detallada de su funcionamiento y se han mostrado los resultados obtenidos. Adicionalmente se han discutido sus posibilidades de implantación industrial, dejando finalmente en entredicho sus posibles aplicaciones pendiente de mejoras de rendimiento del sistema.

El segundo método constituye un desarrollo basado en la inclusión de un acelerómetro en el sistema. Este método, sin embargo, ha permitido obtener unos resultados esclarecedores. Se ha conseguido construir un sistema robusto que permite el control de orientación de forma remota de un dispositivo de una forma estable y precisa. Se ha demostrado que a través de la utilización de plataformas electrónicas básicas como *Arduino* se pueden construir sistemas de regulación completos y autónomos, escalables a un amplio rango de aplicaciones industriales.

Este trabajo multiplataforma demuestra el progreso que se ha seguido a través del descubrimiento de herramientas como *MATLAB* o *Arduino*, y cómo se han explorado sus posibilidades con el fin de cumplir el objetivo último del proyecto, la orientación de forma remota.

Para concluir, cabe señalar que, a pesar de que el diseño se ha dirigido hacia el control de orientación de la antena de detección de descargas parciales, dada su gran versatilidad este método de control es extensible a cualquier proceso que necesite de la orientación remota de un dispositivo.

4.2. Desarrollos futuros

En esta sección se incluyen las diversas posibilidades que se han evaluado de cara a la continuación del desarrollo de este proyecto.

La primera parte de este apartado se dedica a los posibles experimentos y mejoras que podrían ampliar el conocimiento sobre el sistema implementado en el Método 1º (ver sección 3.1). Continuando con los experimentos, habría que comprobar cuál es el efecto que posee la resistencia del circuito *RC* en la sensibilidad y precisión del sensor, a través de la variación de su valor. También se podrían variar otras constantes del sistema, como es el tamaño de la placa del sensor, de nuevo dirigido a aumentar la información sobre el funcionamiento y comportamiento del sistema.

Así mismo, se ha planteado la posibilidad de seguir trabajando con el acelerómetro, ya que es el método que nos ofrece mejores resultados. Dado que la versión del sistema de control evaluada en este trabajo constituye tan sólo un prototipo, el primer paso sería aumentar la integración del sistema con el ser humano. Para ello, el método más común pasa por adherir el acelerómetro a un guante, como muestra la figura 4.1.

Figura 4.1: Acelerómetro integrado en un guante [32]

Además, como se expresó en la sección 2.4.4, se podría aumentar la funcionalidad del sistema utilizando las funciones adicionales del acelerómetro como la interpretación de caída libre o la adición de un umbral a la detección del movimiento.

En esta línea de trabajo, se podría sustituir la placa utilizada (*Arduino* UNO) por una placa *Arduino* LilyPad, ya que ésta última está diseñada para ser utilizada en aplicaciones que incluyan la integración de electrónica en la vestimenta del usuario. Con el objetivo de hacer el sistema más cómodo, también se podría añadir un módulo de comunicación inhalámbrica para eliminar la dependencia que genera la conexión por cables. De esta forma se crearía un sistema de control inhalámbrico alimentado de forma independiente (baterías).

Otra ampliación del proyecto que surge de forma inmediata es la mejora de la interfaz de control en *MATLAB*. Ya se ha implementado un programa que permite visualizar la dirección a la que apuntaría la antena si estuviese acoplada a los servomotores. Sin embargo, existe la posibilidad de mejorar el rendimiento de la misma, disminuyendo las alteraciones que sufren las medidas para obtener un comportamiento más suave. Además, debería permitir la variación por parte del usuario de los parámetros de medición del acelerómetro. Ya que ADXL345 permite tres rangos de sensibilidad de medida: +/- 2g, +/- 4g y +/- 8g; esta interfaz nos permitiría variar este valor, por lo que se podría cambiar la sensibilidad del sistema sin tener que modificar el código de forma manual.

Por último, para dotar de utilidad práctica al prototipo, se debería dotar de movimiento a la antena a controlar (o cualquier otro dispositivo) a través de la construcción o adquisición de un dispositivo como el mostrado en la figura 4.2. Dicho

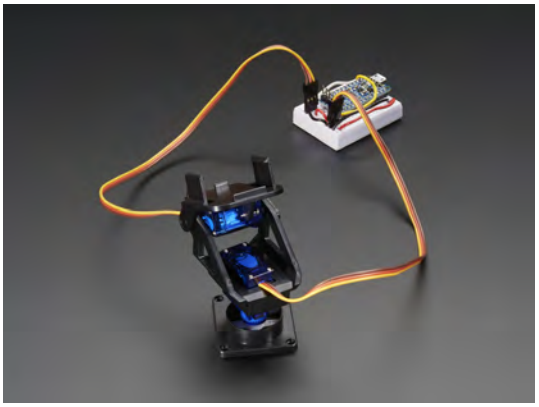

Figura 4.2: Sistema de control por inclinación construido por *Adafruit*

dispositivo consta de dos servomotores y una base móvil. Uno de los servomotores genera el giro de la base en acimut, mientras que el otro realiza la variación de altura. Como se puede observar, este dispositivo de adapta a la perfección a las características del sistema descrito en este trabajo.

Apéndice A

Código

En este apéndice se muestra el código completo de los programas creados durante la realización de este trabajo.

A.1. Código en Arduino

A.1.1. Calibración angular de los servomotores

Para este proceso se utilizan dos programas. En el primero se introducen los ángulos deseados en grados (º), y se obtiene un ángulo de salida:

```
#include <Servo.h> //Añade la librería servo

Servo myServo; //Añade un objeto de tipo 'servo'

int angle; //Esta variable contendrá en ángulo deseado


void setup() {
  myServo.attach(9); //Inicia la comunicación con el servo en el pin digital 9
  myServo.write(0); //Solicita al el servo la posición de 0º

  Serial.begin(9600); //Inicia una comunicación serie con el ordenador

  Serial.println("Introduce a value (0-180): "); //Imprime en el terminal serie la
    frase deseada
}

void loop() {
  if(Serial.available()){

    angle = Serial.parseInt(); //Captura el valor introducido
    myServo.write(angle); //Solicita al servo la posición angular deseada
    }
}
```

Debido a que la función *servo.write()* asume los valores de desplazamiento máximo y mínimo del servo, como si de un servomotor estándar se tratase, es necesario utilizar una función más precisa. Para servomotores no convencionales no se obtendrá la salida deseada.

Por ello se implementó el segundo programa, que permite alcanzar mayor exactitud en las salidas angulares al trabajar directamente con la señal en milisegundos que se envía al servomotor:

```
#include <Servo.h> //Añade la librería servo

Servo myServo; //Añade un objeto de tipo 'servo'

int angle; //Esta variable contendrá en ángulo deseado


void setup() {
  myServo.attach(5); //Inicia la comunicación con el servo en el pin digital 5
  myServo.write(0); //Solicita al el servo la posición de 0º

  Serial.begin(9600); //Inicia una comunicación serie el ordenador

  Serial.println("Introduce a value (ms): "); //Imprime en el terminal serie la
    frase deseada
}

void loop() {
  if(Serial.available() == 0){}

  else{
    angle = Serial.parseInt(); //Captura el valor introducido
    myServo.writeMicroseconds(angle); //Solicita al servo la posición angular
    deseada (en ms)
    }
}
```

Este último programa nos permite, mediante ensayo y error, encontrar los valores angulares deseados mediante la introducción de una posición en milisegundos. Ésto es posible gracias a la capacidad PWM de los puertos digitales de Arduino.

A.1.2. Programa de detección 3D. Método 1º

```
//El programa se basa en un interruptor de milisegundos que se dispara cuando se
    alcanza el valor de refresco
#define resolution 8                //Define la resolución del contador en 8 bits
#define mains 50
#define refresh 2 * 1000000 / mains  //Define la tasa de refresco en microsegundos:
     40000 microsegundos = 40 ms. En binario corresponde al número
    1001110001000000


#include <Servo.h> //Añade la librería servo
Servo servo1; //Añade un objeto de tipo 'servo'
int angle; //Esta variable contendrá en ángulo deseado

void setup() {
  Serial.begin(115200); //Inicia una comunicación serie el ordenador

for(int i = 2; i < 14; i++) { //Configura los puertos de Arduino como salidas
  pinMode(i, OUTPUT);
  digitalWrite(i, LOW); //Pone los puertos a tierra para descargar el condensador
}

for(int i = 8; i < 11; i++) //Configura los pines como entrada
  pinMode(i, INPUT);

//Inicia la función startTimer(), que pone a cero los registros del microprocesador
startTimer();

  servo1.attach(3); //Inicia la comunicación con el servo en el pin digital 3
  servo1.writeMicroseconds(1500); ////Solicita al el servo su posición de punto
    medio (90º)
```

```
27  }
28
29  void loop() {
30
31    angle = map(time(9, B00000010), 9670, 10300, 720, 2300); //Transforma los valores
        recibidos a un rango válido para el servomotor
32
33    //Mueve el servomotor al punto indicado por el sensor
34    servo1.writeMicroseconds(angle);
35    Serial.print(angle);
36    Serial.print(" ");
37    Serial.print(time(8, B00000001), DEC); //B00000001 es el puerto 8 de Arduino.
        Imprime el valor medido por el sensor
38
39  }
40
41  long time(int pin, byte mask) {           //Variable que contiene el tiempo en
      microsegundos que tarda el pin en alcanzar su valor lógico alto
42    unsigned long count = 0, total = 0;
43    while(checkTimer() < refresh) {         //Se cumple mientras el tiempo del reloj
        del microprocesador es inferior a la tasa de refresco
44      pinMode(pin, OUTPUT);
45      PORTB = 0;                            //PORTB sirve para actuar a nivel bajo y
        de forma más rápida sobre el registro de los puertos de Arduino. Selecciona
        los puertos digitales desde el 8 al 13. Esta sentencia los fija en su valor
        lógico bajo
46      pinMode(pin, INPUT);
47      while((PINB & mask) == 0)             //PINB lee los valores de los puertos
        comprendidos entre el 8 y el 13, y mask lee el valor del pin que se desea leer
        , es decir, puertos 8, 9 y 10. Por lo tanto, mientras los puertos leídos sigan
         siendo cero, aumenta el contador
48        count++;
49      total++;
50    }
51    startTimer();                           //Una vez realizada la medida, pone a
        cero los registros temporales del microprocesador
52    return (count << resolution) / total;   //Devuelve el valor del tiempo medido
53  }
54
55  extern volatile unsigned long timer0_overflow_count; //Define el contador de veces
      que el registro se pone a cero como un número se características 'long'
56
57  void startTimer() {
58    //Registros temporales del microprocesador
59    timer0_overflow_count = 0; //Contiene el numero de veces que el contador Timer #0
        se pone a cero
60    TCNT0 = 0; //El contador Timer #0 es un registro de 8 bits que se incrementa una
        unidad cada 0.004 milisegundos (16MHz del microprocesador multiplicado por un
        preescalador de 64). Como es de 8 bits, solo almacena valores hasta 256,
        momento tras el cual se pone a cero. Por tanto, este registro se pone a cero
        cada 1.024 ms.
61  }
62
63  unsigned long checkTimer() {
64    return ((timer0_overflow_count << 8) + TCNT0) << 2;     //Devuelve el tiempo del
        reloj del microprocesador en microsegundos. Contiene las veces que se ha
        puesto a cero el registro, mas el valor actual del contador. Como se puede
        observar en la sentencia de la función, el valor del contador de puesta a cero
         del registro Timer #0 se desplaza 10 bits a la izquierda, lo que corresponde
        al número binario 10000000000 = 1024 en base decimal
65  }
```

A.1.3. Programación del acelerómetro. Método 2º

```
#include <SPI.h> // Incluye la librería SPI para trabajar con el dispositivo maestro

int CS=10; //Asigna el pin SS al pin 10. En ocasiones Slave Select se expresa como Chip Select

#define POWER_CTL 0x2D // Registro de control de energía del acelerómtero
#define DATA_FORMAT 0x31 // Registro del formato de los datos del acelerómetro

unsigned char values[10]; //Este buffer contiene los datos leídos

int x,y,z; //Estos números contienen los valores tomados por el sensor en los ejes X, Y y Z

#include <Servo.h> //Añade la librería servo
Servo servoX; //Añade un objeto de tipo 'servo'
Servo servoY;

int angleX; //Esta variable contendrá en ángulo deseado
int angleY;


void setup(){
  SPI.begin(); // Inicia la comunicación SPI
  SPI.setDataMode(SPI_MODE3); // Configura el bus SPI para comunicarse con el acelerómetro. En este caso se escoge el mode3
  pinMode(CS, OUTPUT); // De nuevo, el dispositivo maestro es el Arduino, por lo que configura el pin SS como salida
  digitalWrite(CS, HIGH); // Fija el pin SS en su valor lógico alto para poder iniciar la comunicación

  writeRegister(DATA_FORMAT, 0x00); // Escribiendo el valor especificado (0x00) en el registro DATA_FORMAT del acelerómetro, lo configura para que tome medidas en un rango de +/- 2g (la dirección de estos registros suele aparecer en el datasheet del dispositivo)

  writeRegister(POWER_CTL, 0x08);  // Inicia el modo 'medición' del acelerómetro escribiendo el valor necesario en el registro de control de energía, POWER_CTL

  //Establece los puertos de control para los servos y los coloca en su punto medio
  servoX.attach(5);
  servoX.writeMicroseconds(1500);
  servoY.attach(6);
  servoY.writeMicroseconds(1400);
}

void loop(){

  //Convierte los valores medidos a valores de trabajo de los servomotores
  angleX = map(x, -255, 255, 720, 2300);
  servoX.writeMicroseconds(angleX);
  angleY = map(y, -255, 255, 550, 2280);
  servoY.writeMicroseconds(angleY);

  // Mediante la lectura de 6 bytes de datos empezando por el registro DATAX0 se obtendrán los valores de la aceleración en los ejes 'x', 'y' y 'z' medidos por el ADXL345
  //Los resultados de la operación de lectura se almacenan en un buffer creado por nosotros: 'values[]'
  readRegister(DATAX0, 6, values);

  //El ADXL345 proporciona valores de aceleración con una resolución de 10 bits (para una configuración de +/- 2g), pero los almacena en 2 bytes (8 bits cada byte). Por lo tanto, se deben combinar dos bytes para obtener la medida completa de cada eje
  //El valor del eje 'x' se almacena en las componentes 0 y 1 del buffer (values[0] y values[1])
```

```
51      x = ((int)values[1]<<8)|(int)values[0];
52      //El valor del eje 'y' se almacena en las componentes 2 y 3 del buffer (values[2]
            y values[3])
53      y = ((int)values[3]<<8)|(int)values[2];
54      //El valor del eje 'x' se almacena en las componentes 4 y 5 del buffer (values[4]
            y values[5])
55      z = ((int)values[5]<<8)|(int)values[4];
56
57      //Muestra los valores medios a través de la comunicación serie
58      Serial.print(x);
59      Serial.print(' ');
60      Serial.println(y);
61
62      //El programa espera 50ms para tomar la siguiente medida. Esto nos permite
            obtener una tasa de refresco de aproximadamente 20Hz.
63      delay(50);
64    }
65
66    //Esta función escribirá las ordenes correspondientes en los registros del ADXL345.
67    //Parámetros:
68    //  char registerAddress - Contiene el registro de destino
69    //  char value - Contiene el valor que se debe escribir en el registro especificado
70    void writeRegister(char registerAddress, char value){
71
72      digitalWrite(CS, LOW);  //Fija el pin SS (chip select) en su valor lógico bajo
            para comenzar la transmisión SPI
73      SPI.transfer(registerAddress);  //Transfiere la dirección del registro de destino
74      SPI.transfer(value);  //Transfiere el valor que se quieren escribir en el
            registro
75      digitalWrite(CS, HIGH); //Fija el pin SS en su valor lógico alto para finalizar
            la transmisión SPI
76    }
77
78    //Esta función leerá un número concreto de registros, empezando por la dirección
            especificada, y almacenará sus valores en un buffer
79    //Parámetros:
80    //  char registerAddress - Contiene la dirección del registro a partir de la cuál
            se comienza la lectura
81    //  int numBytes - Contiene en número de registros que se deben leer
82    //  char * values - Contiene un puntero que indica el buffer en el que se deben
            almacenar los datos
83    void readRegister(char registerAddress, int numBytes, unsigned char * values){
84      //Como se especifica en el Datasheet del ADCL345, para realizar una operación de
            lectura, el bit más significativo de la dirección del registro debe ser fijado
85      char address = 0x80 | registerAddress;
86      //Si se realiza una operación de multi-lectura, el bit número 6 también debe ser
            fijado
87      if(numBytes > 1)address = address | 0x40;
88
89      //Fija el pin SS en su valor lógico bajo para comenzar la transmisión SPI
90      digitalWrite(CS, LOW);
91      //Transfiere la dirección del primer registro que se debe leer
92      SPI.transfer(address);
93      //Continúa leyendo registros hasta que se alcanza el número establecido, y los
            almacena en el buffer designado para ello (values[])
94      for(int i=0; i<numBytes; i++){
95        values[i] = SPI.transfer(0x00);
96      }
97      //Fija el pin SS en su valor lógico alto para finalizar la transmisión SPI
98      digitalWrite(CS, HIGH);
99    }
```

A.2. Código en MATLAB

A.2.1. Calibración angular de los servomotores

Mediante estos programas se comprueba que los valores estimados con *Arduino* son correctos, a la vez que se exploran las posibilidades de Arduino programado con *MATLAB*.

```
ard1 = arduino(); %añade un objeto con las propiedades de Arduino

servo1 = servo(ard1, 3,
    'MinPulseDuration', 720*10^-6,        % Valor mínimo (0°) del servo
    calculada con el programa de Arduino
    'MaxPulseDuration', 2300*10^-6);      % Valor máximo (180°)

angle_real = 0; %Pone el servomotor en su posición de 0°
exit = false; %Esta Variable sirve para detener el programa

while exit == false %se cumple mientras exit = false
angle_real = input('Introduce desired angle (0-180, x = exit): ', 's'); %
    solicita un ángulo en la ventana de comandos

if strcmp(angle_real,'x') %finaliza el programa si exit = true
    exit = true;
else
        %Convierte el valor introducido a un rango que maneja el
           acelerómetro
    angle_real_s = str2double (angle_real);
    angle = angle_real_s/180;

    % Mueve el servo a la posición deseada
    writePosition(servo1, angle);

    % Lee la nueva posición del servo
    current_pos = readPosition(servo1);

    % Muestra la nueva posición en la consola en °
    current_pos = current_pos*180;
    fprintf('Current motor position is %d degrees\n', current_pos);
end

end

clear all
```

A.2.2. Interfaz de comunicación serie de Arduino

Este programa nos permite leer la comunicación serie que *Arduino* establece con el ordenador. Así se podrán extraer los datos medidos por los sensores, y trabajar con ellos en *MATLAB*.

El programa transforma los valores obtenidos de la comunicación serie a coordenadas esféricas y después a cartesianas. Después crea una gráfica que muestra el movimiento virtual de la antena.

```
clear all
clc

H = uicontrol('style','pushbutton','String', 'Stop','Callback',@stopf, 'position',[3 370 100 50], 'UserData', 1); %Código que especifica dónde irá colocado el botón de Stop, el texto que contiene, y la función a la que llama el botón, en este caso, stopf()

p3da = plot3(nan, nan, nan, 'g', 'LineWidth', 5); %Crea una representación con la línea que simula la antena. Como valores se introduce un número no válido, nan, que se actualizará con un bucle para mostrar la posición virtual de la antena en tiempo real
axis ([-1 1 0 1 0 1]); %Establece los límites de los ejes
grid on
view ([135, 35]); %Cambia el ángulo de visión de la gráfica
hold on %Permite insertar nuevos objetos en la representación creada anteriormente, sin la necesidad de crear una representación adicional

%En la gráfica anterior incluye las proyecciones de la antena sobre los planos para facilitar la identificación de su orientación
p3db = plot3(nan, nan, nan, 'b');
axis ([-1 1 0 1 0 1]);
grid on
view ([135, 35]);
hold on

p3dc = plot3(nan, nan, nan, 'r');
axis ([-1 1 0 1 0 1]);
grid on
view ([135, 35]);


arduino = serial('COM3','BaudRate',9600); % Esta variable contiene el puerto y la velocidad de la comunicación serie

fopen(arduino); %Abre la comunicación serie de Arduino


while get(H, 'UserData') == 1 %Se cumple mientras el botón está sin pulsar
    data= fscanf(arduino,'%f %f',[1 2]); %Lee los valores transmitidos a través de la comunicación serie

        %Extrae los valores del vector creado con la lectura de la comunicación serie
    a = data(1, 1) %acimut
    b = data(1, 2) %elevación

    %Ésta función personalizada transforma los valores de entrada a un rango diferente
    azimuth = remap(a, -255, 255, 0, 180) %Esta sentencia mostrará los valores de acimut de la antena en grados
    elevation = remap(b, -255, 255, -90, 90) %Esta sentencia mostrará los valores de elevación de la antena en grados

    %Esta sentencia transforma los datos de salida del acelerómetro a
```

```
    radianes, para poder transformar las coordenadas esféricas en
    cartesianas
42      azimuthr = remap(a, -255, 255, 0, pi);
43      elevationr = remap(b, -255, 255, -pi/2, pi/2);
44
45
46      %Transforma las coordenadas esféricas en coordenadas cartesianas
47      %x=Rcos(el)cos(az)
48      %y=Rcos(el)sin(az)
49      %z=Rsin(el)
50      x = cos(elevationr)*cos(azimuthr);
51      y = cos(elevationr)*sin(azimuthr);
52      z = sin(elevationr);
53
54      %Define los vectores que formarán las líneas que aparecen en la
    representación
55      %Puntos iniciales y finales
56      x1 = 0;
57      y1 = 0;
58      z1 = 0;
59      u1 = x;
60      v1 = y;
61      w1 = z;
62
63      %Vectores
64      X1 = [x1 u1];
65      Y1 = [y1 v1];
66      Z1 = [z1 w1];
67      X2 = [x1 u1];
68      Y2 = [y1 v1];
69      Z2 = [0 0];
70      X3 = [0 0];
71      Y3 = [0 0];
72      Z3 = [z1 w1];
73
74      %Actualiza los valores de la representación para generar mostrar la
    posición en tiempo real
75      set(p3da,'XData',X1);  %Actualiza los datos del eje 'x'
76      set(p3da,'YData',Y1);
77      set(p3da,'ZData',Z1);
78
79      set(p3db,'XData',X2);
80      set(p3db,'YData',Y2);
81      set(p3db,'ZData',Z2);
82
83      set(p3dc,'XData',X3);
84      set(p3dc,'YData',Y3);
85      set(p3dc,'ZData',Z3);
86      drawnow;
87  end
88
89  fclose(arduino); %Cierra la lectura de la comunicación serie
90
91  close all
92  clear all
```

A.2.2.1. Función *remap()*

Ésta función presenta el mismo código que la función *map()* de *Arduino*, por lo que tiene la misma utilidad. Transforma el rango de los valores de entrada, escalándolo mediante los límites mínimo y máximo del rango deseado de salida.

```
function x = remap(value, fromLow, fromHigh, toLow, toHigh)

        %Expresión matemática que transforma los rangos
    x = (value - fromLow) * (toHigh - toLow) / (fromHigh - fromLow) +
    toLow;

end
```

A.2.2.2. Función *stopf()*

Esta función define el funcionamiento del botón 'Stop' de la interfaz de Matlab.

```
function stopf(ObjectH, EventData)
set(ObjectH, 'UserData', 0);
end
```

Apéndice B

Cálculo de errores

En este capítulo se detalla el cálculo de errores de medidas indirectas realizadas a lo largo del proyecto.

B.1. Gravedad

Como se muestra en la página 62, el valor de la gravedad (en g) se calcula de la forma siguiente:

$$a_g = a_m \times Escala = a_m \times 0,0039 \tag{B.1}$$

donde:

- a_g: valor de la gravedad expresado en g (1g $\approx 9{,}81\ \mathrm{m} \times \mathrm{s}^{-2}$)
- a_m: valor de medida del acelerómetro extraído del monitor serie

El error de a_g será igual a:

$$\Delta a_g = \frac{\partial a_g}{\partial a_m} \times \Delta a_m = 0{,}0039 \times 1 = 0{,}039\ \mathrm{g} \tag{B.2}$$

Redondeando a una cifra significativa:

$$\Delta a_g = 0{,}04\ \mathrm{g}$$

B.2. Inclinación

La medida de inclinación se calcula mediante la función *remap()* implementada en *MATLAB*. La expresión de dicha función es la siguiente:

$$a = \frac{(x - fromLow)(toHigh - toLow)}{fromHigh - fromLow} + toLow \tag{B.3}$$

siendo:

- a: valor de salida de la función
- x: valor de entrada de la función

- $fromLow$, $fromHigh$: valores mayor y menor del rango de entrada
- $toLow$, $toHigh$: valores mayor y menor del rango de salida

El error de a será por tanto:

$$\Delta a = \frac{\partial a}{\partial x} \times \Delta x = \frac{(toHigh - toLow)(fromHigh - fromLow)}{(fromHigh - fromLow)^2} \times \Delta x \quad \text{(B.4)}$$

Para los valores utilizados en las mediciones:

$$\Delta a = \frac{(180 - 0)(255 - (-255))}{(255 - (-255))^2} = 0{,}35$$

Redondeando a una cifra significativa:

$$\Delta a = 0{,}4^{\text{o}}$$

B.3. Capacidad

Como expresa la ecuación (9), el tiempo de carga de un condensador se expresa como:

$$t_c = 5\tau = 5RC \quad (9)$$

Operando:

$$C = \frac{t_c}{5R} \quad \text{(B.5)}$$

El error de C es entonces:

$$\Delta C = \frac{\partial C}{\partial t_c} = \frac{1}{5R} \times \Delta t_c \quad \text{(B.6)}$$

Para los valores utilizados en las mediciones:

$$\Delta C = \frac{1}{5 \times 270 \times 10^3} \times 1 \times 10^{-6} = 7{,}4 \times 10^{-13}\text{F}$$

Redondeando a una cifra significativa:

$$\Delta C = 0{,}7 \times 10^{-12}\text{F}$$

Apéndice C

Medidas experimentales

C.1. Método 1

C.1.1. Mano plana

Cuadro C.1: Medidas tomadas con la mano paralela al sensor

Distancia (cm)	**Tiempo (μs)**	1ª medida	2ª medida	3ª medida	**Tiempo medio (μs)**	**Capacidad ($\pm 0,0007$ nF)**
∞	9437	9467	9461	9442	9452	7,0013
10,0	9512	9503	9504	9509	9507	7,0422
9,0	9524	9524	9523	9525	9524	7,0548
8,0	9541	9542	9545	9548	9544	7,0696
7,0	9552	9548	9549	9551	9550	7,0741
6,0	9556	9554	9560	9562	9558	7,0800
5,0	9572	9573	9574	9572	9573	7,0909
4,0	9586	9581	9584	9582	9583	7,0987
3,0	9731	9745	9725	9734	9734	7,2102
2,0	9840	9816	9823	9818	9824	7,2772
1,5	9902	9921	9915	9916	9914	7,3433
1,0	9914	9948	9917	9931	9928	7,3537
0,5	10094	10088	10071	10099	10088	7,4726
0,2	10198	10154	10165	10198	10179	7,5398

C.1.2. Mano a 45º

Cuadro C.2: Medidas tomadas con la mano formando un ángulo de 45º con el sensor

Distancia (cm)	**Tiempo (μs)**	1ª medida	2ª medida	3ª medida	**Tiempo medio (μs)**	**Capacidad ($\pm 0,0007$ nF)**
∞	9437	9461	9456	9440	9449	6,9989
10,0	9512	9509	9517	9515	9513	7,0469
9,0	9517	9508	9509	9518	9513	7,0467
8,0	9512	9513	9517	9522	9516	7,0489
7,0	9527	9524	9527	9516	9524	7,0544
6,0	9539	9526	9538	9539	9536	7,0633
5,0	9541	9543	9552	9543	9545	7,0702
4,0	9556	9558	9557	9548	9555	7,0776
3,0	9585	9585	9586	9572	9582	7,0978
2,0	9633	9632	9633	9634	9633	7,1356
1,5	9664	9665	9663	9664	9664	7,1585
1,0	9754	9750	9752	9764	9755	7,2259
0,5	9815	9801	9810	9820	9812	7,2678
0,2	9931	9958	9952	9993	9959	7,3767

C.1.3. Mano a 90º

Cuadro C.3: Medidas tomadas con la mano formando un ángulo de 90º con el sensor

Distancia (cm)	**Tiempo (μs)**	1ª medida	2ª medida	3ª medida	**Tiempo medio (μs)**	**Capacidad ($\pm 0,0007$ nF)**
∞	9461	9441	9437	9453	9448	6,9985
10,0	9501	9502	9495	9502	9500	7,0370
9,0	9514	9514	9515	9516	9515	7,0480
8,0	9525	9522	9523	9532	9526	7,0559
7,0	9537	9542	9541	9540	9540	7,0667
6,0	9544	9545	9544	9545	9545	7,0700
5,0	9546	9546	9547	9546	9546	7,0713
4,0	9554	9553	9557	9554	9555	7,0774
3,0	9581	9566	9577	9571	9574	7,0917
2,0	9633	9632	9644	9645	9639	7,1396
1,5	9647	9648	9645	9647	9647	7,1457
1,0	9711	9741	9712	9720	9721	7,2007
0,5	9781	9778	9797	9781	9784	7,2476
0,2	9906	9933	9904	9913	9914	7,3437

C.2. Método 2

C.2.1. Giros lentos, suaves y progresivos

Cuadro C.4: Primera toma de medidas

Inclinación real (º)	**Inclinación medida (eje x)** $(\pm 0,4^{o})$	**Valor de salida (eje x)**	**Intensidad de la gravedad** $(\pm 0,04$ **g)**	**Inclinación medida (eje y)** $(\pm 0,4^{o})$	**Valor de salida (eje y)**	**Intensidad de la gravedad** $(\pm 0,04$ **g)**
0,0	6,7	19	0,07	3,9	12	0,05
30,0	30,4	85	0,33	30,7	87	0,34
45,0	45,2	129	0,50	45,9	130	0,51
60,0	60,4	170	0,66	60,4	172	0,67
90,0	100,9	285	1,11	100,3	284	1,11
-30,0	-29,3	-83	-0,32	-29,9	-85	-0,33
-45,0	-45,2	-128	-0,50	-45,5	-129	-0,50
-60,0	-59,6	-168	-0,66	-60,6	-169	-0,66
-90,0	-90,7	-253	-0,99	-91,8	-259	-1,01

Cuadro C.5: Segunda toma de medidas

Inclinación real (º)	**Inclinación medida (eje x)** $(\pm 0,4^{o})$	**Valor de salida (eje x)**	**Intensidad de la gravedad** $(\pm 0,04$ **g)**	**Inclinación medida (eje y)** $(\pm 0,4^{o})$	**Valor de salida (eje y)**	**Intensidad de la gravedad** $(\pm 0,04$ **g)**
0,0	8,1	23	0,09	4,9	14	0,05
30,0	30,1	86	0,34	30,7	86	0,34
45,0	45,5	127	0,50	45,2	128	0,50
60,0	59,3	170	0,66	60,7	171	0,67
90,0	100,3	287	1,12	99,9	285	1,11
-30,0	-30,4	-87	-0,34	-30,6	-85	-0,33
-45,0	-45,5	-129	-0,50	-45,2	-124	-0,48
-60,0	-60,4	-172	-0,67	-60,4	-168	-0,66
-90,0	-89,3	-252	-0,98	-91,4	-260	-1,01

C.2.2. Giros con el eje 'z' en horizontal

Cuadro C.6: Primera toma de medidas

Inclinación referencia (eje x) (º)	**Inclinación medida (eje x)** ($\pm 0,4$º)	**Valor de salida (eje x)**	**Intensidad de la gravedad** ($\pm 0,04$ **g**)	**Inclinación referencia (eje y)** (º)	**Inclinación medida (eje y)** ($\pm 0,4$º)	**Valor de salida (eje y)**	**Intensidad de la gravedad** ($\pm 0,04$ **g**)
90,0	100,3	287	1,12	0,0	0,7	1	0,00
60,0	80,1	224	0,87	30,0	67,8	191	0,74
45,0	60,7	170	0,66	45,0	87,2	245	0,96
30,0	30,6	93	0,36	60,0	98,2	175	0,68
0,0	0,2	2	0,01	90,0	100,3	289	1,13
-90,0	-88,9	-252	-0,98	0,0	-1,1	-1	0,00
-60,0	-82,3	-232	-0,90	-30,0	-30,7	-86	-0,34
-45,0	-73,8	-208	-0,81	-45,0	-46,9	-131	-0,51
-30,0	-50,2	-145	-0,57	-60,0	-70,9	-202	-0,79
0,0	-0,4	-1	0,00	-90,0	-91,1	-258	-1,01

Cuadro C.7: Segunda toma de medidas

Inclinación referencia (eje x) (º)	**Inclinación medida (eje x)** ($\pm 0,4$º)	**Valor de salida (eje x)**	**Intensidad de la gravedad** ($\pm 0,04$ **g**)	**Inclinación referencia (eje y)** (º)	**Inclinación medida (eje y)** ($\pm 0,4$º)	**Valor de salida (eje y)**	**Intensidad de la gravedad** ($\pm 0,04$ **g**)
90,0	101,6	285	1,11	0,0	0,6	2	0,01
60,0	80,2	221	0,86	30,0	67,7	189	0,74
45,0	60,6	172	0,67	45,0	87,8	243	0,95
30,0	30,9	95	0,37	60,0	98,8	178	0,69
0,0	0,4	1	0,00	90,0	100,9	287	1,12
-90,0	-88,8	-253	-0,99	0,0	-2,0	-2	-0,01
-60,0	-83,1	-231	-0,90	-30,0	-31,2	-84	-0,33
-45,0	-73,6	-209	-0,82	-45,0	-46,0	-135	-0,53
-30,0	-50,3	-146	-0,57	-60,0	-70,2	-203	-0,79
0,0	-0,5	-2	-0,01	-90,0	-91,6	-254	-0,99

Apéndice D

Servomotores

D.1. Valores de calibración

A continuación se añaden los valores (para 0º y 180º) estimados mediante ensayo y error con el programa de *Arduino* descrito en la sección 3.1.4.1.

Cuadro D.1: Valores aproximados de calibración de los servomotores

Característica (ms)	Servo (S2309S)	Servo (SANWA)
Anchura mínima de pulso (0º)	550	720
Anchura máxima de pulso (180º)	2280	2300
Punto medio (90º)	1400	1500

D.2. Características de los servomotores

Cuadro D.2: Características de los servomotores utilizados [35]

	SpringRC SM-S2309S	SANWA 1301
Modulación	Analógica	Analógica
Par (kgcm)	1,0 (4,8V)	3,0 (4,8V)
	1,2 (6,0V)	
Velocidad (s/60º)	0,12 (4,8V)	0,2 (4,8V)
	0,12 (6,0V)	
Rango de giro (º)	180	180

Apéndice E

Acelerómetro

A continuación se puede encontrar la hoja de características del acelerómetro.

ANALOG DEVICES

3-Axis, ±2 *g*/±4 *g*/±8 *g*/±16 *g* Digital Accelerometer

ADXL345

FEATURES

- Ultralow power: as low as 40 µA in measurement mode and 0.1 µA in standby mode at V_S = 2.5 V (typical)
- Power consumption scales automatically with bandwidth
- User-selectable resolution
  - Fixed 10-bit resolution
  - Full resolution, where resolution increases with *g* range, up to 13-bit resolution at ±16 *g* (maintaining 4 mg/LSB scale factor in all *g* ranges)
- Embedded, patent pending FIFO technology minimizes host processor load
- Tap/double tap detection
- Activity/inactivity monitoring
- Free-fall detection
- Supply voltage range: 2.0 V to 3.6 V
- I/O voltage range: 1.7 V to V_S
- SPI (3- and 4-wire) and I²C digital interfaces
- Flexible interrupt modes mappable to either interrupt pin
- Measurement ranges selectable via serial command
- Bandwidth selectable via serial command
- Wide temperature range (−40°C to +85°C)
- 10,000 *g* shock survival
- Pb free/RoHS compliant
- Small and thin: 3 mm × 5 mm × 1 mm LGA package

APPLICATIONS

- Handsets
- Medical instrumentation
- Gaming and pointing devices
- Industrial instrumentation
- Personal navigation devices
- Hard disk drive (HDD) protection
- Fitness equipment

GENERAL DESCRIPTION

The ADXL345 is a small, thin, low power, 3-axis accelerometer with high resolution (13-bit) measurement at up to ±16 *g*. Digital output data is formatted as 16-bit twos complement and is accessible through either a SPI (3- or 4-wire) or I²C digital interface.

The ADXL345 is well suited for mobile device applications. It measures the static acceleration of gravity in tilt-sensing applications, as well as dynamic acceleration resulting from motion or shock. Its high resolution (4 mg/LSB) enables measurement of inclination changes less than 1.0°.

Several special sensing functions are provided. Activity and inactivity sensing detect the presence or lack of motion and if the acceleration on any axis exceeds a user-set level. Tap sensing detects single and double taps. Free-fall sensing detects if the device is falling. These functions can be mapped to one of two interrupt output pins. An integrated, patent pending 32-level first in, first out (FIFO) buffer can be used to store data to minimize host processor intervention.

Low power modes enable intelligent motion-based power management with threshold sensing and active acceleration measurement at extremely low power dissipation.

The ADXL345 is supplied in a small, thin, 3 mm × 5 mm × 1 mm, 14-lead, plastic package.

FUNCTIONAL BLOCK DIAGRAM

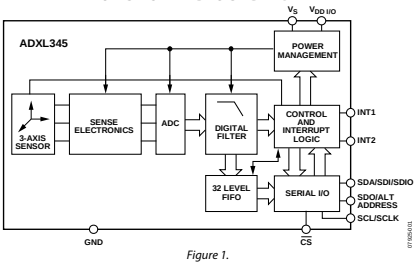

Figure 1.

One Technology Way, P.O. Box 9106, Norwood, MA 02062-9106, U.S.A.
Tel: 781.329.4700 www.analog.com
Fax: 781.461.3113

TABLE OF CONTENTS

REVISION HISTORY

5/09—Revision 0: Initial Version

SPECIFICATIONS

T_A = 25°C, V_S = 2.5 V, $V_{DD\ I/O}$ = 1.8 V, acceleration = 0 *g*, C_S = 1 µF tantalum, C_{IO} = 0.1 µF, unless otherwise noted.

Table 1. Specifications[1]

Parameter	Test Conditions	Min	Typ	Max	Unit
SENSOR INPUT	Each axis				
Measurement Range	User selectable		±2, ±4, ±8, ±16		*g*
Nonlinearity	Percentage of full scale		±0.5		%
Inter-Axis Alignment Error			±0.1		Degrees
Cross-Axis Sensitivity[2]			±1		%
OUTPUT RESOLUTION	Each axis				
All *g* Ranges	10-bit resolution		10		Bits
±2 *g* Range	Full resolution		10		Bits
±4 *g* Range	Full resolution		11		Bits
±8 *g* Range	Full resolution		12		Bits
±16 *g* Range	Full resolution		13		Bits
SENSITIVITY	Each axis				
Sensitivity at X_{OUT}, Y_{OUT}, Z_{OUT}	±2 *g*, 10-bit or full resolution	232	256	286	LSB/*g*
Scale Factor at X_{OUT}, Y_{OUT}, Z_{OUT}	±2 *g*, 10-bit or full resolution	3.5	3.9	4.3	mg/LSB
Sensitivity at X_{OUT}, Y_{OUT}, Z_{OUT}	±4 *g*, 10-bit resolution	116	128	143	LSB/*g*
Scale Factor at X_{OUT}, Y_{OUT}, Z_{OUT}	±4 *g*, 10-bit resolution	7.0	7.8	8.6	mg/LSB
Sensitivity at X_{OUT}, Y_{OUT}, Z_{OUT}	±8 *g*, 10-bit resolution	58	64	71	LSB/*g*
Scale Factor at X_{OUT}, Y_{OUT}, Z_{OUT}	±8 *g*, 10-bit resolution	14.0	15.6	17.2	mg/LSB
Sensitivity at X_{OUT}, Y_{OUT}, Z_{OUT}	±16 *g*, 10-bit resolution	29	32	36	LSB/*g*
Scale Factor at X_{OUT}, Y_{OUT}, Z_{OUT}	±16 *g*, 10-bit resolution	28.1	31.2	34.3	mg/LSB
Sensitivity Change Due to Temperature			±0.01		%/°C
0 *g* BIAS LEVEL	Each axis				
0 *g* Output for X_{OUT}, Y_{OUT}		−150	±40	+150	mg
0 *g* Output for Z_{OUT}		−250	±80	+250	mg
0 *g* Offset vs. Temperature for x-, y-Axes			±0.8		mg/°C
0 *g* Offset vs. Temperature for z-Axis			±4.5		mg/°C
NOISE PERFORMANCE					
Noise (x-, y-Axes)	Data rate = 100 Hz for ±2 *g*, 10-bit or full resolution		<1.0		LSB rms
Noise (z-Axis)	Data rate = 100 Hz for ±2 *g*, 10-bit or full resolution		<1.5		LSB rms
OUTPUT DATA RATE AND BANDWIDTH	User selectable				
Measurement Rate[3]		6.25		3200	Hz
SELF-TEST[4]	Data rate ≥ 100 Hz, 2.0 V ≤ V_S ≤ 3.6 V				
Output Change in x-Axis		0.20		2.10	*g*
Output Change in y-Axis		−2.10		−0.20	*g*
Output Change in z-Axis		0.30		3.40	*g*
POWER SUPPLY					
Operating Voltage Range (V_S)		2.0	2.5	3.6	V
Interface Voltage Range ($V_{DD\ I/O}$)	V_S ≤ 2.5 V	1.7	1.8	V_S	V
	V_S ≥ 2.5 V	2.0	2.5	V_S	V
Supply Current	Data rate > 100 Hz		145		µA
	Data rate < 10 Hz		40		µA
Standby Mode Leakage Current			0.1	2	µA
Turn-On Time[5]	Data rate = 3200 Hz		1.4		ms
TEMPERATURE					
Operating Temperature Range		−40		+85	°C
WEIGHT					
Device Weight			20		mg

[1] All minimum and maximum specifications are guaranteed. Typical specifications are not guaranteed.
[2] Cross-axis sensitivity is defined as coupling between any two axes.
[3] Bandwidth is half the output data rate.
[4] Self-test change is defined as the output (*g*) when the SELF_TEST bit = 1 (in the DATA_FORMAT register) minus the output (*g*) when the SELF_TEST bit = 0 (in the DATA_FORMAT register). Due to device filtering, the output reaches its final value after 4 × τ when enabling or disabling self-test, where τ = 1/(data rate).
[5] Turn-on and wake-up times are determined by the user-defined bandwidth. At a 100 Hz data rate, the turn-on and wake-up times are each approximately 11.1 ms. For other data rates, the turn-on and wake-up times are each approximately τ + 1.1 in milliseconds, where τ = 1/(data rate).

ABSOLUTE MAXIMUM RATINGS

Table 2.

Parameter	Rating
Acceleration	
Any Axis, Unpowered	10,000 *g*
Any Axis, Powered	10,000 *g*
V_S	−0.3 V to +3.6 V
$V_{DD\ I/O}$	−0.3 V to +3.6 V
Digital Pins	−0.3 V to $V_{DD\ I/O}$ + 0.3 V or 3.6 V, whichever is less
All Other Pins	−0.3 V to +3.6 V
Output Short-Circuit Duration (Any Pin to Ground)	Indefinite
Temperature Range	
Powered	−40°C to +105°C
Storage	−40°C to +105°C

Stresses above those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only; functional operation of the device at these or any other conditions above those indicated in the operational section of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

THERMAL RESISTANCE

Table 3. Package Characteristics

Package Type	θ_{JA}	θ_{JC}	Device Weight
14-Terminal LGA	150°C/W	85°C/W	20 mg

ESD CAUTION

ESD (electrostatic discharge) sensitive device. Charged devices and circuit boards can discharge without detection. Although this product features patented or proprietary protection circuitry, damage may occur on devices subjected to high energy ESD. Therefore, proper ESD precautions should be taken to avoid performance degradation or loss of functionality.

PIN CONFIGURATION AND FUNCTION DESCRIPTIONS

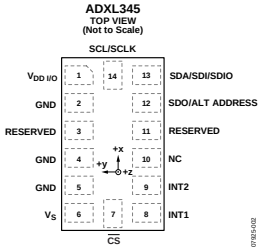

Figure 2. Pin Configuration

Table 4. Pin Function Descriptions

Pin No.	Mnemonic	Description
1	$V_{DD\ I/O}$	Digital Interface Supply Voltage.
2	GND	Must be connected to ground.
3	Reserved	Reserved. This pin must be connected to V_S or left open.
4	GND	Must be connected to ground.
5	GND	Must be connected to ground.
6	V_S	Supply Voltage.
7	$\overline{CS}$	Chip Select.
8	INT1	Interrupt 1 Output.
9	INT2	Interrupt 2 Output.
10	NC	Not Internally Connected.
11	Reserved	Reserved. This pin must be connected to ground or left open.
12	SDO/ALT ADDRESS	Serial Data Output/Alternate I^2C Address Select.
13	SDA/SDI/SDIO	Serial Data (I^2C)/Serial Data Input (SPI 4-Wire)/Serial Data Input and Output (SPI 3-Wire).
14	SCL/SCLK	Serial Communications Clock.

THEORY OF OPERATION

The ADXL345 is a complete 3-axis acceleration measurement system with a selectable measurement range of ±2 *g*, ±4 *g*, ±8 *g*, or ±16 *g*. It measures both dynamic acceleration resulting from motion or shock and static acceleration, such as gravity, which allows the device to be used as a tilt sensor.

The sensor is a polysilicon surface-micromachined structure built on top of a silicon wafer. Polysilicon springs suspend the structure over the surface of the wafer and provide a resistance against acceleration forces.

Deflection of the structure is measured using differential capacitors that consist of independent fixed plates and plates attached to the moving mass. Acceleration deflects the beam and unbalances the differential capacitor, resulting in a sensor output whose amplitude is proportional to acceleration. Phase-sensitive demodulation is used to determine the magnitude and polarity of the acceleration.

POWER SEQUENCING

Power can be applied to V_S or $V_{DD\ I/O}$ in any sequence without damaging the ADXL345. All possible power-on modes are summarized in Table 5. The interface voltage level is set with the interface supply voltage, $V_{DD\ I/O}$, which must be present to ensure that the ADXL345 does not create a conflict on the communication bus. For single-supply operation, $V_{DD\ I/O}$ can be the same as the main supply, V_S. In a dual-supply application, however, $V_{DD\ I/O}$ can differ from V_S to accommodate the desired interface voltage, as long as V_S is greater than $V_{DD\ I/O}$.

After V_S is applied, the device enters standby mode, where power consumption is minimized and the device waits for $V_{DD\ I/O}$ to be applied and for the command to enter measurement mode to be received. (This command can be initiated by setting the measure bit in the POWER_CTL register (Address 0x2D).) In addition, any register can be written to or read from to configure the part while the device is in standby mode. It is recommended to configure the device in standby mode and then to enable measurement mode. Clearing the measure bit returns the device to the standby mode.

Table 5. Power Sequencing

Condition	V_S	$V_{DD\ I/O}$	Description
Power Off	Off	Off	The device is completely off, but there is a potential for a communication bus conflict.
Bus Disabled	On	Off	The device is on in standby mode, but communication is unavailable and will create a conflict on the communication bus. The duration of this state should be minimized during power-up to prevent a conflict.
Bus Enabled	Off	On	No functions are available, but the device will not create a conflict on the communication bus.
Standby or Measurement	On	On	At power-up, the device is in standby mode, awaiting a command to enter measurement mode, and all sensor functions are off. After the device is instructed to enter measurement mode, all sensor functions are available.

POWER SAVINGS

Power Modes

The ADXL345 automatically modulates its power consumption in proportion to its output data rate, as outlined in Table 6. If additional power savings is desired, a lower power mode is available. In this mode, the internal sampling rate is reduced, allowing for power savings in the 12.5 Hz to 400 Hz data rate range but at the expense of slightly greater noise. To enter lower power mode, set the LOW_POWER bit (Bit 4) in the BW_RATE register (Address 0x2C). The current consumption in low power mode is shown in Table 7 for cases where there is an advantage for using low power mode. The current consumption values shown in Table 6 and Table 7 are for a V_S of 2.5 V. Current scales linearly with V_S.

Table 6. Current Consumption vs. Data Rate (T_A = 25°C, V_S = 2.5 V, $V_{DD\ I/O}$ = 1.8 V)

Output Data Rate (Hz)	Bandwidth (Hz)	Rate Code	I_{DD} (μA)
3200	1600	1111	145
1600	800	1110	100
800	400	1101	145
400	200	1100	145
200	100	1011	145
100	50	1010	145
50	25	1001	100
25	12.5	1000	65
12.5	6.25	0111	55
6.25	3.125	0110	40

Table 7. Current Consumption vs. Data Rate, Low Power Mode (T_A = 25°C, V_S = 2.5 V, $V_{DD\ I/O}$ = 1.8 V)

Output Data Rate (Hz)	Bandwidth (Hz)	Rate Code	I_{DD} (μA)
400	200	1100	100
200	100	1011	65
100	50	1010	55
50	25	1001	50
25	12.5	1000	40
12.5	6.25	0111	40

Auto Sleep Mode

Additional power can be saved if the ADXL345 automatically switches to sleep mode during periods of inactivity. To enable this feature, set the THRESH_INACT register (Address 0x25) and the TIME_INACT register (Address 0x26) each to a value that signifies inactivity (the appropriate value depends on the application), and then set the AUTO_SLEEP bit and the link bit in the POWER_CTL register (Address 0x2D). Current consumption at the sub-8 Hz data rates used in this mode is typically 40 μA for a V_S of 2.5 V.

Standby Mode

For even lower power operation, standby mode can be used. In standby mode, current consumption is reduced to 0.1 μA (typical). In this mode, no measurements are made. Standby mode is entered by clearing the measure bit (Bit 3) in the POWER_CTL register (Address 0x2D). Placing the device into standby mode preserves the contents of FIFO.

SERIAL COMMUNICATIONS

I²C and SPI digital communications are available. In both cases, the ADXL345 operates as a slave. I²C mode is enabled if the $\overline{CS}$ pin is tied high to $V_{DD\ I/O}$. The $\overline{CS}$ pin should always be tied high to $V_{DD\ I/O}$ or be driven by an external controller because there is no default mode if the $\overline{CS}$ pin is left unconnected. Therefore, not taking these precautions may result in an inability to communicate with the part. In SPI mode, the $\overline{CS}$ pin is controlled by the bus master. In both SPI and I²C modes of operation, data transmitted from the ADXL345 to the master device should be ignored during writes to the ADXL345.

SPI

For SPI, either 3- or 4-wire configuration is possible, as shown in the connection diagrams in Figure 3 and Figure 4. Clearing the SPI bit in the DATA_FORMAT register (Address 0x31) selects 4-wire mode, whereas setting the SPI bit selects 3-wire mode. The maximum SPI clock speed is 5 MHz with 100 pF maximum loading, and the timing scheme follows clock polarity (CPOL) = 1 and clock phase (CPHA) = 1.

$\overline{CS}$ is the serial port enable line and is controlled by the SPI master. This line must go low at the start of a transmission and high at the end of a transmission, as shown in Figure 5. SCLK is the serial port clock and is supplied by the SPI master. It is stopped high when $\overline{CS}$ is high during a period of no transmission. SDI and SDO are the serial data input and output, respectively. Data should be sampled at the rising edge of SCLK.

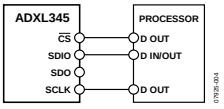

Figure 3. 3-Wire SPI Connection Diagram

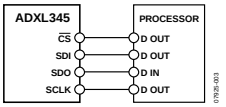

Figure 4. 4-Wire SPI Connection Diagram

To read or write multiple bytes in a single transmission, the multiple-byte bit, located after the R/$\overline{W}$ bit in the first byte transfer (MB in Figure 5 to Figure 7), must be set. After the register addressing and the first byte of data, each subsequent set of clock pulses (eight clock pulses) causes the ADXL345 to point to the next register for a read or write. This shifting continues until the clock pulses cease and $\overline{CS}$ is deasserted. To perform reads or writes on different, nonsequential registers, $\overline{CS}$ must be deasserted between transmissions and the new register must be addressed separately.

The timing diagram for 3-wire SPI reads or writes is shown in Figure 7. The 4-wire equivalents for SPI writes and reads are shown in Figure 5 and Figure 6, respectively.

Table 8. SPI Digital Input/Output Voltage

Parameter	Limit[1]	Unit
Digital Input Voltage		
Low Level Input Voltage (V_{IL})	$0.2 \times V_{DD\ I/O}$	V max
High Level Input Voltage (V_{IH})	$0.8 \times V_{DD\ I/O}$	V min
Digital Output Voltage		
Low Level Output Voltage (V_{OL})	$0.15 \times V_{DD\ I/O}$	V max
High Level Output Voltage (V_{OH})	$0.85 \times V_{DD\ I/O}$	V min

[1] Limits based on characterization results, not production tested.

Table 9. SPI Timing (T_A = 25°C, V_S = 2.5 V, $V_{DD\ I/O}$ = 1.8 V)[1]

	Limit[2, 3]			
Parameter	Min	Max	Unit	Description
f_{SCLK}		5	MHz	SPI clock frequency
t_{SCLK}	200		ns	1/(SPI clock frequency) mark-space ratio for the SCLK input is 40/60 to 60/40
t_{DELAY}	10		ns	$\overline{CS}$ falling edge to SCLK falling edge
t_{QUIET}	10		ns	SCLK rising edge to $\overline{CS}$ rising edge
t_{DIS}		100	ns	$\overline{CS}$ rising edge to SDO disabled
$t_{CS,DIS}$	250		ns	$\overline{CS}$ deassertion between SPI communications
t_S	$0.4 \times t_{SCLK}$		ns	SCLK low pulse width (space)
t_M	$0.4 \times t_{SCLK}$		ns	SCLK high pulse width (mark)
t_{SDO}		95	ns	SCLK falling edge to SDO transition
t_{SETUP}	10		ns	SDI valid before SCLK rising edge
t_{HOLD}	10		ns	SDI valid after SCLK rising edge

[1] The $\overline{CS}$, SCLK, SDI, and SDO pins are not internally pulled up or down; they must be driven for proper operation.
[2] Limits based on characterization results, characterized with f_{SCLK} = 5 MHz and bus load capacitance of 100 pF; not production tested.
[3] The timing values are measured corresponding to the input thresholds (V_{IL} and V_{IH}) given in Table 8.

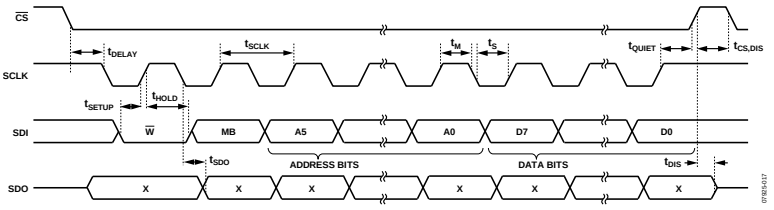

Figure 5. SPI 4-Wire Write

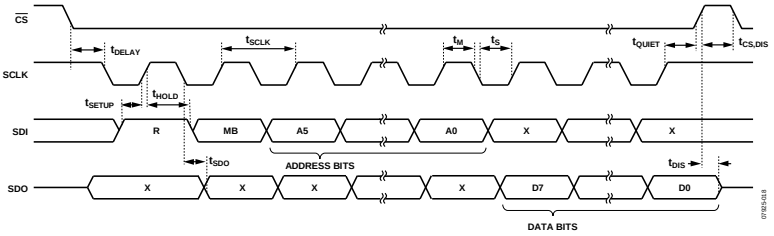

Figure 6. SPI 4-Wire Read

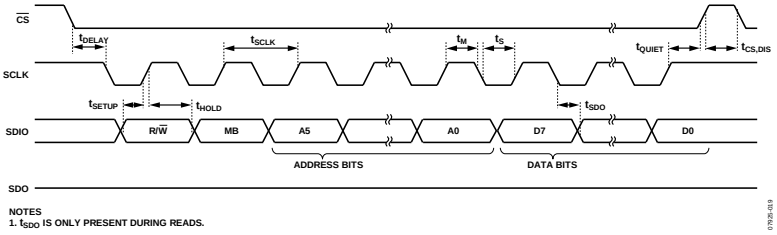

Figure 7. SPI 3-Wire Read/Write

I²C

With $\overline{CS}$ tied high to $V_{DD\ I/O}$, the ADXL345 is in I²C mode, requiring a simple 2-wire connection as shown in Figure 8. The ADXL345 conforms to the *UM10204 I²C-Bus Specification and User Manual*, Rev. 03—19 June 2007, available from NXP Semiconductor. It supports standard (100 kHz) and fast (400 kHz) data transfer modes if the timing parameters given in Table 11 and Figure 10 are met. Single- or multiple-byte reads/writes are supported, as shown in Figure 9. With the SDO/ALT ADDRESS pin high, the 7-bit I²C address for the device is 0x1D, followed by the R/$\overline{W}$ bit. This translates to 0x3A for a write and 0x3B for a read. An alternate I²C address of 0x53 (followed by the R/$\overline{W}$ bit) can be chosen by grounding the SDO/ALT ADDRESS pin (Pin 12). This translates to 0xA6 for a write and 0xA7 for a read.

If other devices are connected to the same I²C bus, the nominal operating voltage level of these other devices cannot exceed $V_{DD\ I/O}$ by more than 0.3 V. External pull-up resistors, R_P, are necessary for proper I²C operation. Refer to the *UM10204 I²C-Bus Specification and User Manual*, Rev. 03—19 June 2007, when selecting pull-up resistor values to ensure proper operation.

Table 10. I²C Digital Input/Output Voltage

Parameter	Limit[1]	Unit
Digital Input Voltage		
Low Level Input Voltage (V_{IL})	$0.25 \times V_{DD\ I/O}$	V max
High Level Input Voltage (V_{IH})	$0.75 \times V_{DD\ I/O}$	V min
Digital Output Voltage		
Low Level Output Voltage (V_{OL})[2]	$0.2 \times V_{DD\ I/O}$	V max

[1] Limits based on characterization results; not production tested.
[2] The limit given is only for $V_{DD\ I/O}$ < 2 V. When $V_{DD\ I/O}$ > 2 V, the limit is 0.4 V max.

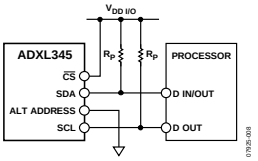

Figure 8. I²C Connection Diagram (Address 0x53)

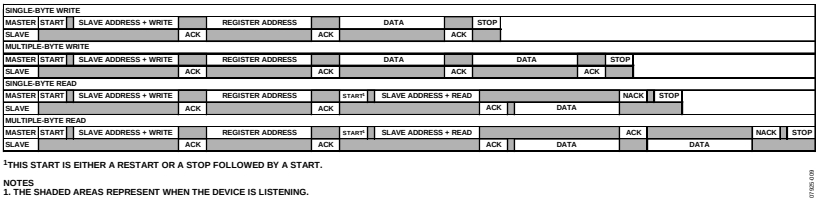

Figure 9. I²C Device Addressing

Table 11. I²C Timing (T_A = 25°C, V_S = 2.5 V, $V_{DD\ I/O}$ = 1.8 V)

	Limit[1, 2]			
Parameter	**Min**	**Max**	**Unit**	**Description**
f_{SCL}		400	kHz	SCL clock frequency
t_1	2.5		µs	SCL cycle time
t_2	0.6		µs	t_{HIGH}, SCL high time
t_3	1.3		µs	t_{LOW}, SCL low time
t_4	0.6		µs	$t_{HD, STA}$, start/repeated start condition hold time
t_5	350		ns	$t_{SU, DAT}$, data setup time
t_6[3, 4, 5, 6]	0	0.65	µs	$t_{HD, DAT}$, data hold time
t_7	0.6		µs	$t_{SU, STA}$, setup time for repeated start
t_8	0.6		µs	$t_{SU, STO}$, stop condition setup time
t_9	1.3		µs	t_{BUF}, bus-free time between a stop condition and a start condition
t_{10}		300	ns	t_R, rise time of both SCL and SDA when receiving
	0		ns	t_R, rise time of both SCL and SDA when receiving or transmitting
t_{11}		250	ns	t_F, fall time of SDA when receiving
		300	ns	t_F, fall time of both SCL and SDA when transmitting
	20 + 0.1 C_b[7]		ns	t_F, fall time of both SCL and SDA when transmitting or receiveing
C_b		400	pF	Capacitive load for each bus line

[1] Limits based on characterization results, with f_{SCL} = 400 kHz and a 3 mA sink current; not production tested.
[2] All values referred to the V_{IH} and the V_{IL} levels given in Table 10.
[3] t_6 is the data hold time that is measured from the falling edge of SCL. It applies to data in transmission and acknowledge times.
[4] A transmitting device must internally provide an output hold time of at least 300 ns for the SDA signal (with respect to $V_{IH(min)}$ of the SCL signal) to bridge the undefined region of the falling edge of SCL.
[5] The maximum t_6 value must be met only if the device does not stretch the low period (t_5) of the SCL signal.
[6] The maximum value for t_6 is a function of the clock low time (t_3), the clock rise time (t_{10}), and the minimum data setup time ($t_{5(min)}$). This value is calculated as $t_{6(max)} = t_3 - t_{10} - t_{5(min)}$.
[7] C_b is the total capacitance of one bus line in picofarads.

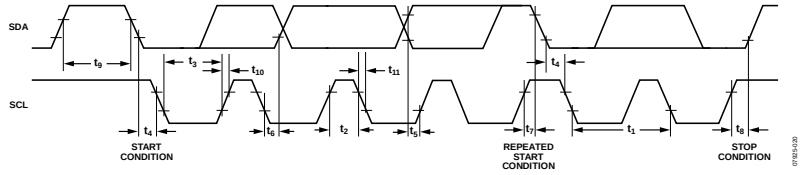

Figure 10. I²C Timing Diagram

INTERRUPTS

The ADXL345 provides two output pins for driving interrupts: INT1 and INT2. Each interrupt function is described in detail in this section. All functions can be used simultaneously, with the only limiting feature being that some functions may need to share interrupt pins. Interrupts are enabled by setting the appropriate bit in the INT_ENABLE register (Address 0x2E) and are mapped to either the INT1 or INT2 pin based on the contents of the INT_MAP register (Address 0x2F). It is recommended that interrupt bits be configured with the interrupts disabled, preventing interrupts from being accidentally triggered during configuration. This can be done by writing a value of 0x00 to the INT_ENABLE register. Clearing interrupts is performed either by reading the data registers (Address 0x32 to Address 0x37) until the interrupt condition is no longer valid for the data-related interrupts or by reading the INT_SOURCE register (Address 0x30) for the remaining interrupts. This section describes the interrupts that can be set in the INT_ENABLE register and monitored in the INT_SOURCE register.

DATA_READY

The DATA_READY bit is set when new data is available and is cleared when no new data is available.

SINGLE_TAP

The SINGLE_TAP bit is set when a single acceleration event that is greater than the value in the THRESH_TAP register (Address 0x1D) occurs for less time than is specified in the DUR register (Address 0x21).

DOUBLE_TAP

The DOUBLE_TAP bit is set when two acceleration events that are greater than the value in the THRESH_TAP register (Address 0x1D) occur for less time than is specified in the DUR register (Address 0x21), with the second tap starting after the time specified by the latent register (Address 0x22) but within the time specified in the window register (Address 0x23). See the Tap Detection section for more details.

Activity

The activity bit is set when acceleration greater than the value stored in the THRESH_ACT register (Address 0x24) is experienced.

Inactivity

The inactivity bit is set when acceleration of less than the value stored in the THRESH_INACT register (Address 0x25) is experienced for more time than is specified in the TIME_INACT register (Address 0x26). The maximum value for TIME_INACT is 255 sec.

FREE_FALL

The FREE_FALL bit is set when acceleration of less than the value stored in the THRESH_FF register (Address 0x28) is experienced for more time than is specified in the TIME_FF register (Address 0x29). The FREE_FALL interrupt differs from the inactivity interrupt as follows: all axes always participate, the timer period is much smaller (1.28 sec maximum), and the mode of operation is always dc-coupled.

Watermark

The watermark bit is set when the number of samples in FIFO equals the value stored in the samples bits (Register FIFO_CTL, Address 0x38). The watermark bit is cleared automatically when FIFO is read, and the content returns to a value below the value stored in the samples bits.

Overrun

The overrun bit is set when new data replaces unread data. The precise operation of the overrun function depends on the FIFO mode. In bypass mode, the overrun bit is set when new data replaces unread data in the DATAX, DATAY, and DATAZ registers (Address 0x32 to Address 0x37). In all other modes, the overrun bit is set when FIFO is filled. The overrun bit is automatically cleared when the contents of FIFO are read.

FIFO

The ADXL345 contains patent pending technology for an embedded 32-level FIFO that can be used to minimize host processor burden. This buffer has four modes: bypass, FIFO, stream, and trigger (see Table 19). Each mode is selected by the settings of the FIFO_MODE bits in the FIFO_CTL register (Address 0x38).

Bypass Mode

In bypass mode, FIFO is not operational and, therefore, remains empty.

FIFO Mode

In FIFO mode, data from measurements of the x-, y-, and z-axes are stored in FIFO. When the number of samples in FIFO equals the level specified in the samples bits of the FIFO_CTL register (Address 0x38), the watermark interrupt is set. FIFO continues accumulating samples until it is full (32 samples from measurements of the x-, y-, and z-axes) and then stops collecting data. After FIFO stops collecting data, the device continues to operate; therefore, features such as tap detection can be used after FIFO is full. The watermark interrupt continues to occur until the number of samples in FIFO is less than the value stored in the samples bits of the FIFO_CTL register.

Stream Mode

In stream mode, data from measurements of the x-, y-, and z-axes are stored in FIFO. When the number of samples in FIFO equals the level specified in the samples bits of the FIFO_CTL register (Address 0x38), the watermark interrupt is set. FIFO continues accumulating samples and holds the latest 32 samples from measurements of the x-, y-, and z-axes, discarding older data as new data arrives. The watermark interrupt continues occurring until the number of samples in FIFO is less than the value stored in the samples bits of the FIFO_CTL register.

Trigger Mode

In trigger mode, FIFO accumulates samples, holding the latest 32 samples from measurements of the x-, y-, and z-axes. After a trigger event occurs and an interrupt is sent to the INT1 or INT2 pin (determined by the trigger bit in the FIFO_CTL register), FIFO keeps the last n samples (where n is the value specified by the samples bits in the FIFO_CTL register) and then operates in FIFO mode, collecting new samples only when FIFO is not full. A delay of at least 5 µs should be present between the trigger event occurring and the start of reading data from the FIFO to allow the FIFO to discard and retain the necessary samples. Additional trigger events cannot be recognized until the trigger mode is reset. To reset the trigger mode, set the device to bypass mode and then set the device back to trigger mode. Note that the FIFO data should be read first because placing the device into bypass mode clears FIFO.

Retrieving Data from FIFO

The FIFO data is read through the DATAX, DATAY, and DATAZ registers (Address 0x32 to Address 0x37). When the FIFO is in FIFO, stream, or trigger mode, reads to the DATAX, DATAY, and DATAZ registers read data stored in the FIFO. Each time data is read from the FIFO, the oldest x-, y-, and z-axes data are placed into the DATAX, DATAY and DATAZ registers.

If a single-byte read operation is performed, the remaining bytes of data for the current FIFO sample are lost. Therefore, all axes of interest should be read in a burst (or multiple-byte) read operation. To ensure that the FIFO has completely popped (that is, that new data has completely moved into the DATAX, DATAY, and DATAZ registers), there must be at least 5 µs between the end of reading the data registers and the start of a new read of the FIFO or a read of the FIFO_STATUS register (Address 0x39). The end of reading a data register is signified by the transition from Register 0x37 to Register 0x38 or by the $\overline{CS}$ pin going high.

For SPI operation at 1.6 MHz or less, the register addressing portion of the transmission is a sufficient delay to ensure that the FIFO has completely popped. For SPI operation greater than 1.6 MHz, it is necessary to deassert the $\overline{CS}$ pin to ensure a total delay of 5 µs; otherwise, the delay will not be sufficient. The total delay necessary for 5 MHz operation is at most 3.4 µs. This is not a concern when using I^2C mode because the communication rate is low enough to ensure a sufficient delay between FIFO reads.

SELF-TEST

The ADXL345 incorporates a self-test feature that effectively tests its mechanical and electronic systems simultaneously. When the self-test function is enabled (via the SELF_TEST bit in the DATA_FORMAT register, Address 0x31), an electrostatic force is exerted on the mechanical sensor. This electrostatic force moves the mechanical sensing element in the same manner as acceleration, and it is additive to the acceleration experienced by the device. This added electrostatic force results in an output change in the x-, y-, and z-axes. Because the electrostatic force is proportional to V_S^2, the output change varies with V_S. The self-test feature of the ADXL345 also exhibits a bimodal behavior that depends on which phase of the clock self-test is enabled. However, the limits shown in Table 1 and Table 12 to Table 15 are valid for all potential self-test values across the entire allowable voltage range. Use of the self-test feature at data rates less than 100 Hz may yield values outside these limits. Therefore, the part should be placed into a data rate of 100 Hz or greater when using self-test.

Table 12. Self-Test Output in LSB for ±2 *g*, Full Resolution

Axis	Min	Max	Unit
X	50	540	LSB
Y	−540	−50	LSB
Z	75	875	LSB

Table 13. Self-Test Output in LSB for ±4 *g*, 10-Bit Resolution

Axis	Min	Max	Unit
X	25	270	LSB
Y	−270	−25	LSB
Z	38	438	LSB

Table 14. Self-Test Output in LSB for ±8 *g*, 10-Bit Resolution

Axis	Min	Max	Unit
X	12	135	LSB
Y	−135	−12	LSB
Z	19	219	LSB

Table 15. Self-Test Output in LSB for ±16 *g*, 10-Bit Resolution

Axis	Min	Max	Unit
X	6	67	LSB
Y	−67	−6	LSB
Z	10	110	LSB

REGISTER MAP

Table 16. Register Map

Address					
Hex	**Dec**	**Name**	**Type**	**Reset Value**	**Description**
0x00	0	DEVID	R	11100101	Device ID.
0x01 to 0x01C	1 to 28	Reserved			Reserved. Do not access.
0x1D	29	THRESH_TAP	R/$\overline{W}$	00000000	Tap threshold.
0x1E	30	OFSX	R/$\overline{W}$	00000000	X-axis offset.
0x1F	31	OFSY	R/$\overline{W}$	00000000	Y-axis offset.
0x20	32	OFSZ	R/$\overline{W}$	00000000	Z-axis offset.
0x21	33	DUR	R/$\overline{W}$	00000000	Tap duration.
0x22	34	Latent	R/$\overline{W}$	00000000	Tap latency.
0x23	35	Window	R/$\overline{W}$	00000000	Tap window.
0x24	36	THRESH_ACT	R/$\overline{W}$	00000000	Activity threshold.
0x25	37	THRESH_INACT	R/$\overline{W}$	00000000	Inactivity threshold.
0x26	38	TIME_INACT	R/$\overline{W}$	00000000	Inactivity time.
0x27	39	ACT_INACT_CTL	R/$\overline{W}$	00000000	Axis enable control for activity and inactivity detection.
0x28	40	THRESH_FF	R/$\overline{W}$	00000000	Free-fall threshold.
0x29	41	TIME_FF	R/$\overline{W}$	00000000	Free-fall time.
0x2A	42	TAP_AXES	R/$\overline{W}$	00000000	Axis control for tap/double tap.
0x2B	43	ACT_TAP_STATUS	R	00000000	Source of tap/double tap.
0x2C	44	BW_RATE	R/$\overline{W}$	00001010	Data rate and power mode control.
0x2D	45	POWER_CTL	R/$\overline{W}$	00000000	Power-saving features control.
0x2E	46	INT_ENABLE	R/$\overline{W}$	00000000	Interrupt enable control.
0x2F	47	INT_MAP	R/$\overline{W}$	00000000	Interrupt mapping control.
0x30	48	INT_SOURCE	R	00000010	Source of interrupts.
0x31	49	DATA_FORMAT	R/$\overline{W}$	00000000	Data format control.
0x32	50	DATAX0	R	00000000	X-Axis Data 0.
0x33	51	DATAX1	R	00000000	X-Axis Data 1.
0x34	52	DATAY0	R	00000000	Y-Axis Data 0.
0x35	53	DATAY1	R	00000000	Y-Axis Data 1.
0x36	54	DATAZ0	R	00000000	Z-Axis Data 0.
0x37	55	DATAZ1	R	00000000	Z-Axis Data 1.
0x38	56	FIFO_CTL	R/$\overline{W}$	00000000	FIFO control.
0x39	57	FIFO_STATUS	R	00000000	FIFO status.

REGISTER DEFINITIONS

Register 0x00—DEVID (Read Only)

D7	D6	D5	D4	D3	D2	D1	D0
1	1	1	0	0	1	0	1

The DEVID register holds a fixed device ID code of 0xE5 (345 octal).

Register 0x1D—THRESH_TAP (Read/Write)

The THRESH_TAP register is eight bits and holds the threshold value for tap interrupts. The data format is unsigned, so the magnitude of the tap event is compared with the value in THRESH_TAP. The scale factor is 62.5 mg/LSB (that is, 0xFF = +16 *g*). A value of 0 may result in undesirable behavior if tap/double tap interrupts are enabled.

Register 0x1E, Register 0x1F, Register 0x20—OFSX, OFSY, OFSZ (Read/Write)

The OFSX, OFSY, and OFSZ registers are each eight bits and offer user-set offset adjustments in twos complement format with a scale factor of 15.6 mg/LSB (that is, 0x7F = +2 *g*).

Register 0x21—DUR (Read/Write)

The DUR register is eight bits and contains an unsigned time value representing the maximum time that an event must be above the THRESH_TAP threshold to qualify as a tap event. The scale factor is 625 µs/LSB. A value of 0 disables the tap/double tap functions.

Register 0x22—Latent (Read/Write)

The latent register is eight bits and contains an unsigned time value representing the wait time from the detection of a tap event to the start of the time window (defined by the window register) during which a possible second tap event can be detected. The scale factor is 1.25 ms/LSB. A value of 0 disables the double tap function.

Register 0x23—Window (Read/Write)

The window register is eight bits and contains an unsigned time value representing the amount of time after the expiration of the latency time (determined by the latent register) during which a second valid tap can begin. The scale factor is 1.25 ms/LSB. A value of 0 disables the double tap function.

Register 0x24—THRESH_ACT (Read/Write)

The THRESH_ACT register is eight bits and holds the threshold value for detecting activity. The data format is unsigned, so the magnitude of the activity event is compared with the value in the THRESH_ACT register. The scale factor is 62.5 mg/LSB. A value of 0 may result in undesirable behavior if the activity interrupt is enabled.

Register 0x25—THRESH_INACT (Read/Write)

The THRESH_INACT register is eight bits and holds the threshold value for detecting inactivity. The data format is unsigned, so the magnitude of the inactivity event is compared with the value in the THRESH_INACT register. The scale factor is 62.5 mg/LSB. A value of 0 mg may result in undesirable behavior if the inactivity interrupt is enabled.

Register 0x26—TIME_INACT (Read/Write)

The TIME_INACT register is eight bits and contains an unsigned time value representing the amount of time that acceleration must be less than the value in the THRESH_INACT register for inactivity to be declared. The scale factor is 1 sec/LSB. Unlike the other interrupt functions, which use unfiltered data (see the Threshold section), the inactivity function uses filtered output data. At least one output sample must be generated for the inactivity interrupt to be triggered. This results in the function appearing unresponsive if the TIME_INACT register is set to a value less than the time constant of the output data rate. A value of 0 results in an interrupt when the output data is less than the value in the THRESH_INACT register.

Register 0x27—ACT_INACT_CTL (Read/Write)

D7	D6	D5	D4
ACT ac/dc	ACT_X enable	ACT_Y enable	ACT_Z enable
D3	**D2**	**D1**	**D0**
INACT ac/dc	INACT_X enable	INACT_Y enable	INACT_Z enable

ACT AC/DC and INACT AC/DC Bits

A setting of 0 selects dc-coupled operation, and a setting of 1 enables ac-coupled operation. In dc-coupled operation, the current acceleration magnitude is compared directly with THRESH_ACT and THRESH_INACT to determine whether activity or inactivity is detected.

In ac-coupled operation for activity detection, the acceleration value at the start of activity detection is taken as a reference value. New samples of acceleration are then compared to this reference value, and if the magnitude of the difference exceeds the THRESH_ACT value, the device triggers an activity interrupt.

Similarly, in ac-coupled operation for inactivity detection, a reference value is used for comparison and is updated whenever the device exceeds the inactivity threshold. After the reference value is selected, the device compares the magnitude of the difference between the reference value and the current acceleration with THRESH_INACT. If the difference is less than the value in THRESH_INACT for the time in TIME_INACT, the device is considered inactive and the inactivity interrupt is triggered.

ACT_x Enable Bits and INACT_x Enable Bits

A setting of 1 enables x-, y-, or z-axis participation in detecting activity or inactivity. A setting of 0 excludes the selected axis from participation. If all axes are excluded, the function is disabled.

Register 0x28—THRESH_FF (Read/Write)

The THRESH_FF register is eight bits and holds the threshold value, in unsigned format, for free-fall detection. The root-sum-square (RSS) value of all axes is calculated and compared with the value in THRESH_FF to determine if a free-fall event occurred. The scale factor is 62.5 mg/LSB. Note that a value of 0 mg may result in undesirable behavior if the free-fall interrupt is enabled. Values between 300 mg and 600 mg (0x05 to 0x09) are recommended.

Register 0x29—TIME_FF (Read/Write)

The TIME_FF register is eight bits and stores an unsigned time value representing the minimum time that the RSS value of all axes must be less than THRESH_FF to generate a free-fall interrupt. The scale factor is 5 ms/LSB. A value of 0 may result in undesirable behavior if the free-fall interrupt is enabled. Values between 100 ms and 350 ms (0x14 to 0x46) are recommended.

Register 0x2A—TAP_AXES (Read/Write)

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	Suppress	TAP_X enable	TAP_Y enable	TAP_Z enable

Suppress Bit

Setting the suppress bit suppresses double tap detection if acceleration greater than the value in THRESH_TAP is present between taps. See the Tap Detection section for more details.

TAP_x Enable Bits

A setting of 1 in the TAP_X enable, TAP_Y enable, or TAP_Z enable bit enables x-, y-, or z-axis participation in tap detection. A setting of 0 excludes the selected axis from participation in tap detection.

Register 0x2B—ACT_TAP_STATUS (Read Only)

D7	D6	D5	D4	D3	D2	D1	D0
0	ACT_X source	ACT_Y source	ACT_Z source	Asleep	TAP_X source	TAP_Y source	TAP_Z source

ACT_x Source and TAP_x Source Bits

These bits indicate the first axis involved in a tap or activity event. A setting of 1 corresponds to involvement in the event, and a setting of 0 corresponds to no involvement. When new data is available, these bits are not cleared but are overwritten by the new data. The ACT_TAP_STATUS register should be read before clearing the interrupt. Disabling an axis from participation clears the corresponding source bit when the next activity or tap/double tap event occurs.

Asleep Bit

A setting of 1 in the asleep bit indicates that the part is asleep, and a setting of 0 indicates that the part is not asleep. See the Register 0x2D—POWER_CTL (Read/Write) section for more information on autosleep mode.

Register 0x2C—BW_RATE (Read/Write)

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	LOW_POWER	Rate			

LOW_POWER Bit

A setting of 0 in the LOW_POWER bit selects normal operation, and a setting of 1 selects reduced power operation, which has somewhat higher noise (see the Power Modes section for details).

Rate Bits

These bits select the device bandwidth and output data rate (see Table 6 and Table 7 for details). The default value is 0x0A, which translates to a 100 Hz output data rate. An output data rate should be selected that is appropriate for the communication protocol and frequency selected. Selecting too high of an output data rate with a low communication speed results in samples being discarded.

Register 0x2D—POWER_CTL (Read/Write)

D7	D6	D5	D4	D3	D2	D1	D0
0	0	Link	AUTO_SLEEP	Measure	Sleep	Wakeup	

Link Bit

A setting of 1 in the link bit with both the activity and inactivity functions enabled delays the start of the activity function until inactivity is detected. After activity is detected, inactivity detection begins, preventing the detection of activity. This bit serially links the activity and inactivity functions. When this bit is set to 0, the inactivity and activity functions are concurrent. Additional information can be found in the Link Mode section.

When clearing the link bit, it is recommended that the part be placed into standby mode and then set back to measurement mode with a subsequent write. This is done to ensure that the device is properly biased if sleep mode is manually disabled; otherwise, the first few samples of data after the link bit is cleared may have additional noise, especially if the device was asleep when the bit was cleared.

AUTO_SLEEP Bit

If the link bit is set, a setting of 1 in the AUTO_SLEEP bit sets the ADXL345 to switch to sleep mode when inactivity is detected (that is, when acceleration has been below the THRESH_INACT value for at least the time indicated by TIME_INACT). A setting of 0 disables automatic switching to sleep mode. See the description of the sleep bit in this section for more information.

When clearing the AUTO_SLEEP bit, it is recommended that the part be placed into standby mode and then set back to measurement mode with a subsequent write. This is done to ensure that the device is properly biased if sleep mode is manually disabled; otherwise, the first few samples of data after the AUTO_SLEEP bit is cleared may have additional noise, especially if the device was asleep when the bit was cleared.

Measure Bit

A setting of 0 in the measure bit places the part into standby mode, and a setting of 1 places the part into measurement mode. The ADXL345 powers up in standby mode with minimum power consumption.

Sleep Bit

A setting of 0 in the sleep bit puts the part into the normal mode of operation, and a setting of 1 places the part into sleep mode. Sleep mode suppresses DATA_READY, stops transmission of data to FIFO, and switches the sampling rate to one specified by the wakeup bits. In sleep mode, only the activity function can be used.

When clearing the sleep bit, it is recommended that the part be placed into standby mode and then set back to measurement mode with a subsequent write. This is done to ensure that the device is properly biased if sleep mode is manually disabled; otherwise, the first few samples of data after the sleep bit is cleared may have additional noise, especially if the device was asleep when the bit was cleared.

Wakeup Bits

These bits control the frequency of readings in sleep mode as described in Table 17.

Table 17. Frequency of Readings in Sleep Mode

Setting		
D1	**D0**	**Frequency (Hz)**
0	0	8
0	1	4
1	0	2
1	1	1

Register 0x2E—INT_ENABLE (Read/Write)

D7	**D6**	**D5**	**D4**
DATA_READY	SINGLE_TAP	DOUBLE_TAP	Activity
D3	**D2**	**D1**	**D0**
Inactivity	FREE_FALL	Watermark	Overrun

Setting bits in this register to a value of 1 enables their respective functions to generate interrupts, whereas a value of 0 prevents the functions from generating interrupts. The DATA_READY, watermark, and overrun bits enable only the interrupt output; the functions are always enabled. It is recommended that interrupts be configured before enabling their outputs.

Register 0x2F—INT_MAP (Read/Write)

D7	**D6**	**D5**	**D4**
DATA_READY	SINGLE_TAP	DOUBLE_TAP	Activity
D3	**D2**	**D1**	**D0**
Inactivity	FREE_FALL	Watermark	Overrun

Any bits set to 0 in this register send their respective interrupts to the INT1 pin, whereas bits set to 1 send their respective interrupts to the INT2 pin. All selected interrupts for a given pin are ORed.

Register 0x30—INT_SOURCE (Read Only)

D7	**D6**	**D5**	**D4**
DATA_READY	SINGLE_TAP	DOUBLE_TAP	Activity
D3	**D2**	**D1**	**D0**
Inactivity	FREE_FALL	Watermark	Overrun

Bits set to 1 in this register indicate that their respective functions have triggered an event, whereas a value of 0 indicates that the corresponding event has not occurred. The DATA_READY, watermark, and overrun bits are always set if the corresponding events occur, regardless of the INT_ENABLE register settings, and are cleared by reading data from the DATAX, DATAY, and DATAZ registers. The DATA_READY and watermark bits may require multiple reads, as indicated in the FIFO mode descriptions in the FIFO section. Other bits, and the corresponding interrupts, are cleared by reading the INT_SOURCE register.

Register 0x31—DATA_FORMAT (Read/Write)

D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
SELF_TEST	SPI	INT_INVERT	0	FULL_RES	Justify	Range	

The DATA_FORMAT register controls the presentation of data to Register 0x32 through Register 0x37. All data, except that for the ±16 *g* range, must be clipped to avoid rollover.

SELF_TEST Bit

A setting of 1 in the SELF_TEST bit applies a self-test force to the sensor, causing a shift in the output data. A value of 0 disables the self-test force.

SPI Bit

A value of 1 in the SPI bit sets the device to 3-wire SPI mode, and a value of 0 sets the device to 4-wire SPI mode.

INT_INVERT Bit

A value of 0 in the INT_INVERT bit sets the interrupts to active high, and a value of 1 sets the interrupts to active low.

FULL_RES Bit

When this bit is set to a value of 1, the device is in full resolution mode, where the output resolution increases with the *g* range set by the range bits to maintain a 4 mg/LSB scale factor. When the FULL_RES bit is set to 0, the device is in 10-bit mode, and the range bits determine the maximum *g* range and scale factor.

Justify Bit

A setting of 1 in the justify bit selects left (MSB) justified mode, and a setting of 0 selects right justified mode with sign extension.

Range Bits

These bits set the *g* range as described in Table 18.

Table 18. *g* Range Setting

Setting		
D1	**D0**	***g* Range**
0	0	±2 *g*
0	1	±4 *g*
1	0	±8 *g*
1	1	±16 *g*

Register 0x32 to Register 0x37—DATAX0, DATAX1, DATAY0, DATAY1, DATAZ0, DATAZ1 (Read Only)

These six bytes (Register 0x32 to Register 0x37) are eight bits each and hold the output data for each axis. Register 0x32 and Register 0x33 hold the output data for the x-axis, Register 0x34 and Register 0x35 hold the output data for the y-axis, and Register 0x36 and Register 0x37 hold the output data for the z-axis. The output data is twos complement, with DATAx0 as the least significant byte and DATAx1 as the most significant byte, where x represent X, Y, or Z. The DATA_FORMAT register (Address 0x31) controls the format of the data. It is recommended that a multiple-byte read of all registers be performed to prevent a change in data between reads of sequential registers.

Register 0x38—FIFO_CTL (Read/Write)

D7	D6	D5	D4	D3	D2	D1	D0
FIFO_MODE		Trigger	Samples				

FIFO_MODE Bits

These bits set the FIFO mode, as described in Table 19.

Table 19. FIFO Modes

Setting			
D7	**D6**	**Mode**	**Function**
0	0	Bypass	FIFO is bypassed.
0	1	FIFO	FIFO collects up to 32 values and then stops collecting data, collecting new data only when FIFO is not full.
1	0	Stream	FIFO holds the last 32 data values. When FIFO is full, the oldest data is overwritten with newer data.
1	1	Trigger	When triggered by the trigger bit, FIFO holds the last data samples before the trigger event and then continues to collect data until full. New data is collected only when FIFO is not full.

Trigger Bit

A value of 0 in the trigger bit links the trigger event of trigger mode to INT1, and a value of 1 links the trigger event to INT2.

Samples Bits

The function of these bits depends on the FIFO mode selected (see Table 20). Entering a value of 0 in the samples bits immediately sets the watermark status bit in the INT_SOURCE register, regardless of which FIFO mode is selected. Undesirable operation may occur if a value of 0 is used for the samples bits when trigger mode is used.

Table 20. Samples Bits Functions

FIFO Mode	Samples Bits Function
Bypass	None.
FIFO	Specifies how many FIFO entries are needed to trigger a watermark interrupt.
Stream	Specifies how many FIFO entries are needed to trigger a watermark interrupt.
Trigger	Specifies how many FIFO samples are retained in the FIFO buffer before a trigger event.

0x39—FIFO_STATUS (Read Only)

D7	D6	D5	D4	D3	D2	D1	D0
FIFO_TRIG	0	Entries					

FIFO_TRIG Bit

A 1 in the FIFO_TRIG bit corresponds to a trigger event occurring, and a 0 means that a FIFO trigger event has not occurred.

Entries Bits

These bits report how many data values are stored in FIFO. Access to collect the data from FIFO is provided through the DATAX, DATAY, and DATAZ registers. FIFO reads must be done in burst or multiple-byte mode because each FIFO level is cleared after any read (single- or multiple-byte) of FIFO. FIFO stores a maximum of 32 entries, which equates to a maximum of 33 entries available at any given time because an additional entry is available at the output filter of the device.

APPLICATIONS INFORMATION

POWER SUPPLY DECOUPLING

A 1 µF tantalum capacitor (C_S) at V_S and a 0.1 µF ceramic capacitor (C_{IO}) at $V_{DD\ I/O}$ placed close to the ADXL345 supply pins is used for testing and is recommended to adequately decouple the accelerometer from noise on the power supply. If additional decoupling is necessary, a resistor or ferrite bead, no larger than 100 Ω, in series with V_S may be helpful. Additionally, increasing the bypass capacitance on V_S to a 10 µF tantalum capacitor in parallel with a 0.1 µF ceramic capacitor may also improve noise.

Care should be taken to ensure that the connection from the ADXL345 ground to the power supply ground has low impedance because noise transmitted through ground has an effect similar to noise transmitted through V_S. It is recommended that V_S and $V_{DD\ I/O}$ be separate supplies to minimize digital clocking noise on the V_S supply. If this is not possible, additional filtering of the supplies as previously mentioned may be necessary.

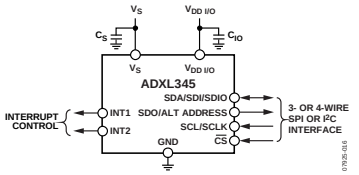

Figure 11. Application Diagram

MECHANICAL CONSIDERATIONS FOR MOUNTING

The ADXL345 should be mounted on the PCB in a location close to a hard mounting point of the PCB to the case. Mounting the ADXL345 at an unsupported PCB location, as shown in Figure 12, may result in large, apparent measurement errors due to undampened PCB vibration. Locating the accelerometer near a hard mounting point ensures that any PCB vibration at the accelerometer is above the accelerometer's mechanical sensor resonant frequency and, therefore, effectively invisible to the accelerometer.

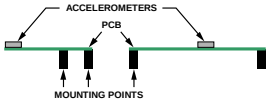

Figure 12. Incorrectly Placed Accelerometers

TAP DETECTION

The tap interrupt function is capable of detecting either single or double taps. The following parameters are shown in Figure 13 for a valid single and valid double tap event:

- The tap detection threshold is defined by the THRESH_TAP register (Address 0x1D).
- The maximum tap duration time is defined by the DUR register (Address 0x21).
- The tap latency time is defined by the latent register (Address 0x22) and is the waiting period from the end of the first tap until the start of the time window, when a second tap can be detected, which is determined by the value in the window register (Address 0x23).
- The interval after the latency time (set by the latent register) is defined by the window register. Although a second tap must begin after the latency time has expired, it need not finish before the end of the time defined by the window register.

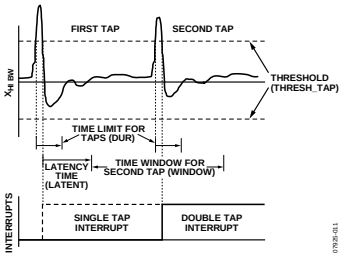

Figure 13. Tap Interrupt Function with Valid Single and Double Taps

If only the single tap function is in use, the single tap interrupt is triggered when the acceleration goes below the threshold, as long as DUR has not been exceeded. If both single and double tap functions are in use, the single tap interrupt is triggered when the double tap event has been either validated or invalidated.

Several events can occur to invalidate the second tap of a double tap event. First, if the suppress bit in the TAP_AXES register (Address 0x2A) is set, any acceleration spike above the threshold during the latency time (set by the latent register) invalidates the double tap detection, as shown in Figure 14.

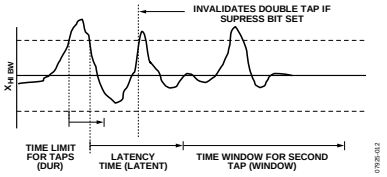

Figure 14. Double Tap Event Invalid Due to High g Event When the Suppress Bit Is Set

A double tap event can also be invalidated if acceleration above the threshold is detected at the start of the time window for the second tap (set by the window register). This results in an invalid double tap at the start of this window, as shown in Figure 15. Additionally, a double tap event can be invalidated if an accel-

eration exceeds the time limit for taps (set by the DUR register), resulting in an invalid double tap at the end of the DUR time limit for the second tap event, also shown in Figure 15.

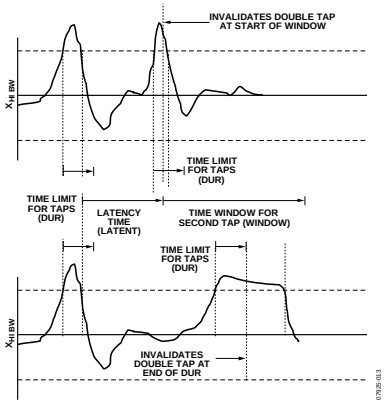

Figure 15. Tap Interrupt Function with Invalid Double Taps

Single taps, double taps, or both can be detected by setting the respective bits in the INT_ENABLE register (Address 0x2E). Control over participation of each of the three axes in single tap/double tap detection is exerted by setting the appropriate bits in the TAP_AXES register (Address 0x2A). For the double tap function to operate, both the latent and window registers must be set to a nonzero value.

Every mechanical system has somewhat different single tap/double tap responses based on the mechanical characteristics of the system. Therefore, some experimentation with values for the latent, window, and THRESH_TAP registers is required. In general, a good starting point is to set the latent register to a value greater than 0x10, to set the window register to a value greater than 0x10, and to set the THRESH_TAP register to be greater than 3 *g*. Setting a very low value in the latent, window, or THRESH_TAP register may result in an unpredictable response due to the accelerometer picking up echoes of the tap inputs.

After a tap interrupt has been received, the first axis to exceed the THRESH_TAP level is reported in the ACT_TAP_STATUS register (Address 0x2B). This register is never cleared, but is overwritten with new data.

THRESHOLD

The lower output data rates are achieved by decimating a common sampling frequency inside the device. The activity, free-fall, and single tap/double tap detection functions are performed using unfiltered data. Since the output data is filtered, the high frequency and high *g* data that is used to determine activity, free-fall, and single tap/double tap events may not be present if the output of the accelerometer is examined. This may result in trigger events being detected when acceleration does not appear to trigger an event because the unfiltered data may have exceeded a threshold or remained below a threshold for a certain period of time while the filtered output data has not exceeded such a threshold.

LINK MODE

The function of the link bit is to reduce the number of activity interrupts that the processor must service by setting the device to look for activity only after inactivity. For proper operation of this feature, the processor must still respond to the activity and inactivity interrupts by reading the INT_SOURCE register (Address 0x30) and, therefore, clearing the interrupts. If an activity interrupt is not cleared, the part cannot go into autosleep mode. The asleep bit in the ACT_TAP_STATUS register (Address 0x2B) indicates if the part is asleep.

SLEEP MODE VS. LOW POWER MODE

In applications where a low data rate is sufficient and low power consumption is desired, it is recommended that the low power mode be used in conjunction with the FIFO. The sleep mode, while offering a low data rate and low average current consumption, suppresses the DATA_READY interrupt, preventing the accelerometer from sending an interrupt signal to the host processor when data is ready to be collected. In this application, setting the part into low power mode (by setting the LOW_POWER bit in the BW_RATE register) and enabling the FIFO in FIFO mode to collect a large value of samples reduces the power consumption of the ADXL345 and allows the host processor to go to sleep while the FIFO is filling up.

USING SELF-TEST

The self-test change is defined as the difference between the acceleration output of an axis with self-test enabled and the acceleration output of the same axis with self-test disabled (see Endnote 4 of Table 1). This definition assumes that the sensor does not move between these two measurements, because if the sensor moves, a non–self-test related shift corrupts the test.

Proper configuration of the ADXL345 is also necessary for an accurate self-test measurement. The part should be set with a data rate greater than or equal to 100 Hz. This is done by ensuring that a value greater than or equal to 0x0A is written into the rate bits (Bit D3 through Bit D0) in the BW_RATE register (Address 0x2C). It is also recommended that the part be set to full-resolution, 16 *g* mode to ensure that there is sufficient dynamic range for the entire self-test shift. This is done by setting Bit D3 of the DATA_FORMAT register (Address 0x31) and writing a value of 0x03 to the range bits (Bit D1 and Bit D0) of the DATA_FORMAT register (Address 0x31). This results in a high dynamic range for measurement and a 3.9 mg/LSB scale factor.

After the part is configured for accurate self-test measurement, several samples of x-, y-, and z-axis acceleration data should be retrieved from the sensor and averaged together. The number of

samples averaged is a choice of the system designer, but a recommended starting point is 0.1 sec worth of data, which corresponds to 10 samples at 100 Hz data rate. The averaged values should be stored and labeled appropriately as the self-test disabled data, that is, X_{ST_OFF}, Y_{ST_OFF}, and Z_{ST_OFF}.

Next, self-test should be enabled by setting Bit D7 of the DATA_FORMAT register (Address 0x31). The output needs some time (about four samples) to settle after enabling self-test. After allowing the output to settle, several samples of the x-, y-, and z-axis acceleration data should be taken again and averaged. It is recommended that the same number of samples be taken for this average as was previously taken. These averaged values should again be stored and labeled appropriately as the value with self-test enabled, that is, X_{ST_ON}, Y_{ST_ON}, and Z_{ST_ON}. Self-test can then be disabled by clearing Bit D7 of the DATA_FORMAT register (Address 0x31).

With the stored values for self-test enabled and disabled, the self-test change is as follows:

$$X_{ST} = X_{ST\_ON} - X_{ST\_OFF}$$

$$Y_{ST} = Y_{ST\_ON} - Y_{ST\_OFF}$$

$$Z_{ST} = Z_{ST\_ON} - Z_{ST\_OFF}$$

Because the measured output for each axis is expressed in LSBs, X_{ST}, Y_{ST}, and Z_{ST} are also expressed in LSBs. These values can be converted to *g*'s of acceleration by multiplying each value by the 3.9 mg/LSB scale factor, if configured for full-resolution, 16 *g* mode. Additionally, Table 12 through Table 15 correspond to the self-test range converted to LSBs and can be compared with the measured self-test change. If the part was placed into full-resolution, 16 *g* mode, the values listed in Table 12 should be used. Although the fixed 10-bit mode or a range other than 16 *g* can be used, a different set of values, as indicated in Table 13 through Table 15, would need to be used. Using a range below 8 *g* may result in insufficient dynamic range and should be considered when selecting the range of operation for measuring self-test. In addition, note that the range in Table 1 and the values in Table 12 through Table 15 take into account all possible supply voltages, V_S, and no additional conversion due to V_S is necessary.

If the self-test change is within the valid range, the test is considered successful. Generally, a part is considered to pass if the minimum magnitude of change is achieved. However, a part that changes by more than the maximum magnitude is not necessarily a failure.

AXES OF ACCELERATION SENSITIVITY

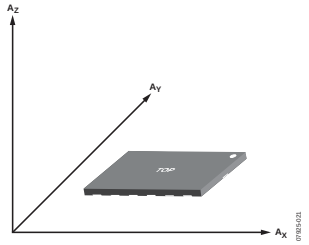

Figure 16. Axes of Acceleration Sensitivity (Corresponding Output Voltage Increases When Accelerated Along the Sensitive Axis)

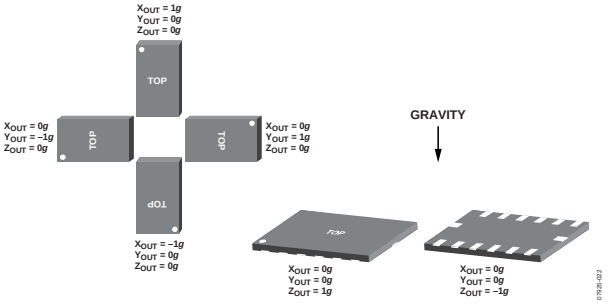

Figure 17. Output Response vs. Orientation to Gravity

LAYOUT AND DESIGN RECOMMENDATIONS

Figure 18 shows the recommended printed wiring board land pattern. Figure 19 and Table 21 provide details about the recommended soldering profile.

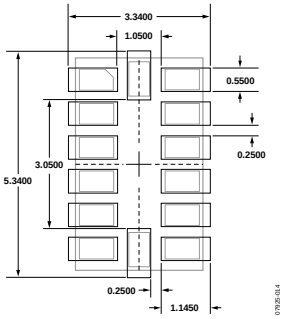

Figure 18. Recommended Printed Wiring Board Land Pattern (Dimensions shown in millimeters)

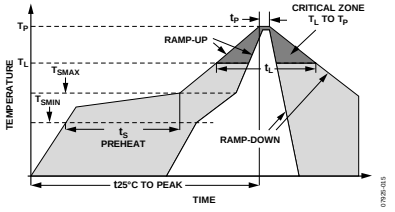

Figure 19. Recommended Soldering Profile

Table 21. Recommended Soldering Profile[1, 2]

	Condition	
Profile Feature	**Sn63/Pb37**	**Pb-Free**
Average Ramp Rate from Liquid Temperature (T_L) to Peak Temperature (T_P)	3°C/sec max	3°C/sec max
Preheat		
Minimum Temperature (T_{SMIN})	100°C	150°C
Maximum Temperature (T_{SMAX})	150°C	200°C
Time from T_{SMIN} to T_{SMAX} (t_S)	60 sec to 120 sec	60 sec to 180 sec
T_{SMAX} to T_L Ramp-Up Rate	3°C/sec max	3°C/sec max
Liquid Temperature (T_L)	183°C	217°C
Time Maintained Above T_L (t_L)	60 sec to 150 sec	60 sec to 150 sec
Peak Temperature (T_P)	240 + 0/−5°C	260 + 0/−5°C
Time of Actual T_P − 5°C (t_P)	10 sec to 30 sec	20 sec to 40 sec
Ramp-Down Rate	6°C/sec max	6°C/sec max
Time 25°C to Peak Temperature	6 minutes max	8 minutes max

[1] Based on JEDEC Standard J-STD-020D.1.
[2] For best results, the soldering profile should be in accordance with the recommendations of the manufacturer of the solder paste used.

OUTLINE DIMENSIONS

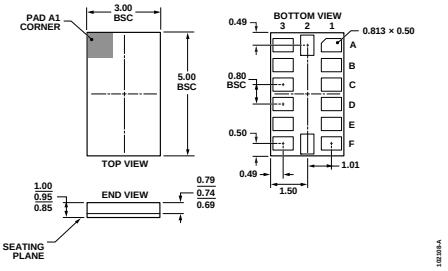

Figure 20. 14-Terminal Land Grid Array [LGA]
(CC-14-1)
Solder Terminations Finish Is Au over Ni
(Dimensions shown in millimeters)

ORDERING GUIDE

Model	Measurement Range (*g*)	Specified Voltage (V)	Temperature Range	Package Description	Package Option
ADXL345BCCZ[1]	±2, ±4, ±8, ±16	2.5	−40°C to +85°C	14-Terminal Land Grid Array [LGA]	CC-14-1
ADXL345BCCZ-RL[1]	±2, ±4, ±8, ±16	2.5	−40°C to +85°C	14-Terminal Land Grid Array [LGA]	CC-14-1
ADXL345BCCZ-RL7[1]	±2, ±4, ±8, ±16	2.5	−40°C to +85°C	14- Terminal Land Grid Array [LGA]	CC-14-1
EVAL-ADXL345Z[1]				Evaluation Board	
EVAL-ADXL345Z-M[1]				Analog Devices Inertial Sensor Evaluation System, Includes ADXL345 Satellite	
EVAL-ADXL345Z-S[1]				ADXL345 Satellite, Standalone	

[1] Z = RoHS Compliant Part.

Bibliografía

[1] ppl4world. (2012, Julio) Ardusat - your arduino experiment in space. [Online]. Disponible: https://www.kickstarter.com/projects/575960623/ardusat-your-arduino-experiment-in-space/description

[2] A. Akl, C. Feng, and S. Valaee, "A novel accelerometer-based gesture recognition system," *IEEE Transactions on Signal Processing*, vol. 59, no. 12, pp. 6197–6205, 2011.

[3] L. Camarinha-Matos and I. ebrary, *Technological innovation for sustainability: second IFIP WG 5.5/SOCOLNET doctoral conference on computing, electrical and industrial systems, DoCEIS 2011, Costa de Caparica, Portugal, February 21-23, 2011, proceedings*. Heidelberg; Berlin: IFIP International Federation for Information Processing, 2011, pp. 216, 217.

[4] M. I. of Technology. (2010, Mayo) Gesture-based computing on the cheap. [Online]. Disponible: http://phys.org/news/2010-05-gesture-based-cheap-video.html

[5] D. W. Goebl. (2008) Motion capture of piano performance. institute of music acoustics (iwk). [Online]. Disponible: http://iwk.mdw.ac.at/goebl/pianomocap.html

[6] Y. Makino, Y. Makino, S. Ogawa, S. Ogawa, H. Shinoda, and H. Shinoda, "Flexible emg sensor array for haptic interface." IEEE, 2008, pp. 1468 – 1473.

[7] (2015) Arduino homepage. [Online]. Disponible: http://www.arduino.cc/

[8] J. Nussey and I. Books24x7, *Arduino for dummies*. Chichester: John Wiley, 2013.

[9] M. Banzi, *Getting Started with Arduino, 2nd Edition*. Maker Media, Inc, 2011.

[10] D. Wheat, *Arduino Internals*. Berkeley, CA: Apress, 2012.

[11] M. McRoberts, *Beginning Arduino, Second Edition*. Apress, 2013.

[12] R. Anderson, D. Cervo, I. ebrary, and I. Books24x7, *Pro Arduino*. Berkeley, CA: Apress, 2013.

[13] J. Blum, I. ebrary, and I. Books24x7, *Exploring Arduino: tools and techniques for engineering wizardry*. Indianapolis, Ind: Wiley, 2013.

[14] J. J. Noble, *Programming interactivity: a designer´s guide to processing, Arduino and openFramworks*. Sebastopol, Calif: O'Reilly, 2009.

[15] (2015) Matlab. [Online]. Disponible: http://es.mathworks.com/products/matlab/

[16] A. Hughes, B. Drury, and I. Books24x7, *Electric motors and drives: fundamentals, types, and applications, fourth edition*. Waltham, Mass; Kidlington, Oxford: Newnes, 2013.

[17] P. E. Sandin, *Robot mechanisms and mechanical devices illustrated*. New York [etc.]: McGraw-Hill, 2003.

[18] S. Ghosh, *Electrical Machines, Second Edition*. Pearson India, 2012.

[19] E. Sazonov and M. R. Neuman, *Wearable Sensors*. Academic Press, 2014.

[20] J. Doscher. (2015) Accelerometer design and applications. [Online]. Disponible: http://iespuigcastellar.xeill.net/Members/vcarceler/articulos/jugando-con-el-wiimote-y-gnu-linux/sensor971.pdf

[21] B. A. Jones, R. Reese, and J. W. Bruce, *Microcontrollers, Second Edition*. Course Technology PTR, 2014.

[22] M.-C. Hsieh, Y.-H. Yen, and T.-Y. Sun, "Gesture recognition with two 3-axis accelerometers," in *Consumer Electronics-Taiwan (ICCE-TW), 2014 IEEE International Conference on*. IEEE, 2014, pp. 239 – 240.

[23] S. Kurosu, S. Kurosu, and T. Yamazaki, "Gyroscopic force measuring system: theory and applications," vol. 1. IEEE, 2004, pp. 825–830 vol. 1.

[24] Sparkfun triple axis accelerometer breakout - adxl345. [Online]. Disponible: https://www.sparkfun.com/products/9836

[25] Adxl345 digital accelerometer. [Online]. Disponible: https://learn.adafruit.com/adxl345-digital-accelerometer/overview

[26] A. Y. A. J.O. Dennis, M.S.B. Mat Sihat and F. Ahmad, "Piezoresistive pressure sensor design, simulation and modification using coventor ware software," *Journal of Applied Sciences*, vol. 11, pp. 1426–1430, 2011.

[27] kylemcdonald. Diy 3d controller. [Online]. Disponible: http://www.instructables.com/id/DIY-3D-Controller/

[28] C. Rawlins, *Basic AC Circuits, 2nd Edition*. Newnes, 2000.

[29] K. S. Kumar, *Electric Circuit Analysis*. Pearson India, 2013.

[30] J. A. Svoboda and R. C. Dorf, *Introduction to Electric Circuits, 9th Edition*. John Wiley & Sons, 2013.

[31] A. S. Morris, *Principios de mediciones e instrumentacion*. Mexico [etc.]: Pearson Educacion, 2002.

[32] M. Ramilli. (2010) Mouseglove. [Online]. Disponible: http://mouseglove.sourceforge.net/

[33] Rc charging circuit. [Online]. Disponible: https://www.google.es/webhp?sourceid=chrome-instant&ion=1&espv=2&ie=UTF-8#q=ex%C3%A1ctamente

[34] Pull up resistor / pull down resistor. [Online]. Disponible: http://www.resistorguide.com/pull-up-resistor_pull-down-resistor/

[35] Servodatabase.com. [Online]. Disponible: http://www.servodatabase.com/